# Table of Contents

WebSphere Application Server Performance Cookbook......................................................................8
  Introduction........................................................................................................................8
  Authors.............................................................................................................................8
  Contributors..................................................................................................................9
  General Performance Recipe..........................................................................................9
  Aspects of Performance Tuning.......................................................................................10
  Load Testing.....................................................................................................................26

Operating Systems..............................................................................................................29
  Central Processing Unit (CPU).........................................................................................29
  Random Access Memory (RAM), Physical Memory.......................................................32
  Input/Output (I/O). .........................................................................................................33
  Network Interface Cards (NICs) and Switches...............................................................34
  TCP/IP............................................................................................................................34
  Clocks..............................................................................................................................40
  Ulimits.............................................................................................................................40
  Linux...............................................................................................................................43
  AIX...................................................................................................................................46
  z/OS..................................................................................................................................95
  IBM i...............................................................................................................................137
  Windows........................................................................................................................143
  Solaris.............................................................................................................................163
  HP-UX.............................................................................................................................174
  macOS............................................................................................................................179

Java......................................................................................................................................181
  Java Recipe.....................................................................................................................181
  General..........................................................................................................................182
  Garbage Collection.......................................................................................................182
  InetAddress Cache.........................................................................................................190
  32-bit versus 64-bit.......................................................................................................190
  Synchronization and Lock Contention.........................................................................191
  Deadlocks......................................................................................................................192
  Classloading..................................................................................................................192
  Explicit Garbage Collection (System.gc, Runtime.gc).................................................193
  java.nio.DirectByteBuffer.........................................................................................194
  Reflection Inflation......................................................................................................195
  Serviceability...............................................................................................................195
  IBM Java Runtime Environment..................................................................................195
  Oracle Java Runtime Environment..............................................................................222

Java Profilers..................................................................................................................231
  Java Profilers Recipe.....................................................................................................232
  Statistical/Sampling Profilers.......................................................................................232
  Tracing Profilers...........................................................................................................242
  Performance Inspector.................................................................................................246

WebSphere Application Server..........................................................................................246
<table>
<thead>
<tr>
<th>Recipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Database</td>
</tr>
<tr>
<td>Tibero Database</td>
</tr>
<tr>
<td>pureScale</td>
</tr>
<tr>
<td>DB2</td>
</tr>
<tr>
<td>Caching and WebSphere eXtreme Scale</td>
</tr>
<tr>
<td>Caching Recipes</td>
</tr>
<tr>
<td>General Caching Topics</td>
</tr>
<tr>
<td>WebSphere eXtreme Scale (WXS)</td>
</tr>
<tr>
<td>WebSphere MQ</td>
</tr>
<tr>
<td>WMQ versus WAS SIB</td>
</tr>
<tr>
<td>WAS Considerations</td>
</tr>
<tr>
<td>Best Practices</td>
</tr>
<tr>
<td>WMQ Versions</td>
</tr>
<tr>
<td>Windows and UNIX Performance Tuning</td>
</tr>
<tr>
<td>WMQ JMS Client</td>
</tr>
<tr>
<td>Resources</td>
</tr>
<tr>
<td>WebSphere DataPower</td>
</tr>
<tr>
<td>WebSphere Commerce</td>
</tr>
<tr>
<td>Tuning</td>
</tr>
<tr>
<td>Caching</td>
</tr>
<tr>
<td>Database connection</td>
</tr>
<tr>
<td>Redbooks</td>
</tr>
<tr>
<td>Troubleshooting</td>
</tr>
<tr>
<td>WebSphere Portal</td>
</tr>
<tr>
<td>WebSphere ESB</td>
</tr>
<tr>
<td>Processing Large Objects</td>
</tr>
<tr>
<td>Aggregation Design Patterns</td>
</tr>
<tr>
<td>Asynchronous Invocation of Synchronous Services Design Patterns</td>
</tr>
<tr>
<td>Shared Libraries</td>
</tr>
<tr>
<td>Parsing Modes</td>
</tr>
<tr>
<td>Memory Analyzer Plugin</td>
</tr>
<tr>
<td>Comparative Transformation Technologies (XSLT vs. Business Object Map)</td>
</tr>
<tr>
<td>First Messing Response Time</td>
</tr>
<tr>
<td>Restricting the Instances of Mediation Flows on the JVM Heap</td>
</tr>
<tr>
<td>Throttling Individual Applications</td>
</tr>
<tr>
<td>Defining a Timeout on Synchronous JAX-WS Imports</td>
</tr>
<tr>
<td>Best Practices and Tuning Red Papers</td>
</tr>
<tr>
<td>Authentication</td>
</tr>
<tr>
<td>Lightweight Directory Access Protocol (LDAP)</td>
</tr>
<tr>
<td>LdapQuery.ear</td>
</tr>
<tr>
<td>IBM Tivoli Directory Server (TDS)</td>
</tr>
<tr>
<td>OpenLDAP</td>
</tr>
<tr>
<td>Competition and Migration</td>
</tr>
<tr>
<td>Comparing Products</td>
</tr>
<tr>
<td>WAS Migration Performance Differences</td>
</tr>
<tr>
<td>Recipes</td>
</tr>
<tr>
<td>Contents</td>
</tr>
</tbody>
</table>
WebSphere Application Server Performance Cookbook

DON'T PANIC
The Hitchhiker's Guide to the Galaxy, Douglas Adams

Great, kid. Don't get cocky.
Han Solo, Star Wars

Introduction

The WebSphere Application Server Performance Cookbook covers performance tuning for WebSphere Application Server (WAS), although there is also a very strong focus on Java, Operating Systems, and theory which can be applied to other products and environments. The cookbook is designed to be read in a few different ways:

1. On the go: Readers short on time should skip to the Recipes chapter at the end of the book. In the spirit of a cookbook, there are recipes that provide step-by-step instructions of how to gather and analyze data for particular classes of problems.
2. General areas: For readers interested in tuning some general area such as WAS or Java, each major chapter provides its recipe at the top of the chapter that summarizes the key tuning knobs that should be investigated.
3. Deep dive: Readers interested in end-to-end tuning are encouraged to skim the entire book for areas relevant to their product usage.

In general, this book is not intended to be read end-to-end. A large portion of the cookbook is more of a reference book. The nature of performance tuning is that 80% of the time, you should focus on a few key things, but 20% of the time you may need to deep dive into a very specific component.

The high level topics covered in the book in depth are: Operating Systems, Java, WebSphere Application Server, Web Servers and more.

Note: Before using this information, read the Notices section for information on terms of use, statement of support, trademarks, etc.

A public developerWorks forum exists for feedback and discussion: https://www.ibm.com/developerworks/community/forums/html/forum?id=e76b33e9-07b8-4bb1-a8d1-ee672a3402e8

Copyright International Business Machines Corporation 2018. All rights reserved. U.S. Government Users Restricted Rights: Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corporation.

Authors

• Alexandre Polozoff (polozoff@us.ibm.com) [Primary Contact]
Contributors

- Andrew Levandoski
- Benedict Fernandes
- Chris Bailey
- Pavel Malyutin
- Rengan Sundararaman
- Stephen A Hellberg
- Tom Alcott
- Walt Adams

Thank you to the IBM managers that helped support this project: Dave Schell, Melissa Modjeski, Keri Olson, Frank Schembari, Michael Stuy, Mike Morin, Sree Ratnasinghe, and others.

General Performance Recipe

Every major section of this cookbook will have a recipe summarizing the key points. The following recipe is for the whole cookbook:

1. Performance tuning is usually about focusing on a few key variables. We will highlight the most common tuning knobs that can often improve the speed of the average application by 200% or more relative to the default configuration. The first step, however, should be to use and be guided by the tools and methodologies. Gather data, analyze it and create hypotheses: then test your hypotheses. Rinse and repeat. As Donald Knuth says: "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say
about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified. It is often a mistake to make a priori judgments about what parts of a program are really critical, since the universal experience of programmers who have been using measurement tools has been that their intuitive guesses fail." (Donald Knuth, Structured Programming with go to Statements, Stanford University, 1974, Association for Computing Machinery)

2. There is a seemingly daunting number of tuning knobs. We try to document everything in detail in case you hit a problem in that area; however, unless you are trying to squeeze out every last drop of performance, we do not recommend a close study of every point.

3. In general, we advocate a bottom-up approach. For example, with a typical WebSphere Application Server application, start with the operating system, then Java, then WAS, then the application, etc. Ideally, investigate these at the same time. The main goal of a performance tuning exercise is to iteratively determine the bottleneck restricting response times and throughput. For example, investigate operating system CPU and memory usage, followed by Java garbage collection usage and/or thread dumps/sampling profilers, followed by WAS PMI, etc. See the Major Tools chapter for recommendations on tools to use in each case.

4. One of the most difficult aspects of performance tuning is understanding whether or not the architecture of the system, or even the test itself, is valid and/or optimal.

5. Meticulously describe and track the problem, each test and its results.

6. Use basic statistics (minimums, maximums, averages, medians, and standard deviations) instead of spot observations.

7. When benchmarking, use a repeatable test that accurately models production behavior, and avoid short term benchmarks which may not have time to warm up.

8. Take the time to automate as much as possible: not just the testing itself, but also data gathering and analysis. This will help you iterate and test more hypotheses.

9. Make sure you are using the latest version of every product because there are often performance or tooling improvements available.

10. When researching problems, you can either analyze or isolate them. Analyzing means taking particular symptoms and generating hypotheses on how to change those symptoms. Isolating means eliminating issues singly until you've discovered important facts. In general, we have found through experience that analysis is preferable to isolation.

11. Review the full end-to-end architecture. Certain internal or external products, devices, content delivery networks, etc. may artificially limit throughput (e.g. Denial of Service protection), periodically mark services down (e.g. network load balancers, WAS plugin, etc.), or become saturated themselves (e.g. CPU on load balancers, etc.).

**Aspects of Performance Tuning**

**Why does performance matter?**

A 2008 survey of 160 organizations (average revenue $1.3 billion) found a typical impact of a one second delay in response times for web applications entailed a potential annual revenue loss of $117 million, 11% fewer page views, 7% fewer conversions, 16% lower customer satisfaction, brand damage, more support calls, and increased costs (Customers are Won or Lost in One Second, Aberdeen
Other benefits include reduced hardware needs and reduced costs, reduced maintenance, reduced power consumption, knowing your breaking points, accurate system sizing, etc.

"Increased performance can often involve sacrificing a certain level of feature or function in the application or the application server. The tradeoff between performance and feature must be weighed carefully when evaluating performance tuning changes."

A typical performance exercise can yield a throughput improvement of about 200% relative to default tuning parameters.

**Basic Definitions**

In general, the goal of performance tuning is to increase throughput, reduce response times, and increase the capacity for concurrent requests, all balanced against costs.

- A response time is the time taken to complete a unit of work. For example, the time taken to complete an HTTP response.
- Concurrent requests is the count of requests processing at the same time. For example, the number of HTTP requests concurrently being processed. A single user may send multiple concurrent requests.
- Throughput is the number of successful responses over a period of time; for example, successful HTTP responses per second. Throughput is proportional to response times and concurrent requests. When throughput saturates, response times will increase.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the buckle zone, one of the system components becomes exhausted. At this point, throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the web server exhaust the limits of the network adapter or if the requests exceed operating system limits for file
A hypothesis is a testable idea. It is not believed to be true nor false. A theory is the result of testing a hypothesis and getting a positive result. It is believed to be true.

**Bottlenecks**

Consider the following methods to eliminate a bottleneck (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_troubleshoot.html):

- Reduce the demand
- Increase resources
- Improve workload distribution
- Reduce synchronization

Reducing the demand for resources can be accomplished in several ways. Caching can greatly reduce the use of system resources by returning a previously cached response, thereby avoiding the work needed to construct the original response. Caching is supported at several points in the following systems:

- IBM HTTP Server
- Command
- Enterprise bean
- Operating system

Application code profiling can lead to a reduction in the CPU demand by pointing out hot spots you can optimize. IBM Rational and other companies have tools to perform code profiling. An analysis of the application might reveal areas where some work might be reduced for some types of transactions.

Change tuning parameters to increase some resources, for example, the number of file handles, while other resources might need a hardware change, for example, more or faster CPUs, or additional application servers. Key tuning parameters are described for each major WebSphere Application Server component to facilitate solving performance problems. Also, the performance advisors page can provide advice on tuning a production system under a real or simulated load.

Workload distribution can affect performance when some resources are underutilized and others are overloaded. WebSphere Application Server workload management functions provide several ways to determine how the work is distributed. Workload distribution applies to both a single server and configurations with multiple servers and nodes.

Some critical sections of the application and server code require synchronization to prevent multiple threads from running this code simultaneously and leading to incorrect results. Synchronization preserves correctness, but it can also reduce throughput when several threads must wait for one thread to exit the critical section. When several threads are waiting to enter a critical section, a thread dump
shows these threads waiting in the same procedure. Synchronization can often be reduced by: changing the code to only use synchronization when necessary; reducing the path length of the synchronized code; or reducing the frequency of invoking the synchronized code.

**Architecture/Clustering**

It is always important to consider what happens when some part of a cluster crashes. Will the rest of the cluster handle it gracefully? Does the heap size have enough head room? Is there enough CPU to handle extra load, etc.? If there is more traffic than the cluster can handle, will it queue and timeout gracefully?

**Methodology**

Begin by understanding that one cannot solve all problems immediately. We recommend prioritizing work into short term (high), 3 months (medium) and long term (low). How the work is prioritized depends on the business requirements and where the most pain is being felt.

Guide yourself primarily with tools and methodologies. Gather data, analyze it, create hypotheses, and test your hypotheses. Rinse and repeat. In general, we advocate a bottom-up approach. For example, with a typical WebSphere Application Server application, start with the operating system, then Java, then WAS, then the application, etc.

The following are some example scenarios and approaches (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_troubleshoot.html). They are specific to particular products and symptoms and we just want to highlight them to give you a taste of how to do performance tuning. Later chapters will go through the details.

**Scenario: Poor performance occurs with only a single user.**

Suggested solution: Utilize request metrics to determine how much each component is contributing to the overall response time. Focus on the component accounting for the most time. Use the Tivoli Performance Viewer to check for resource consumption, including frequency of garbage collections. You might need code profiling tools to isolate the problem to a specific method.

**Scenario: Poor performance only occurs with multiple users.**

Suggested solution: Check to determine if any systems have high CPU, network or disk utilization and address those. For clustered configurations, check for uneven loading across cluster members.

**Scenario: None of the systems seems to have a CPU, memory, network, or disk constraint but performance problems occur with multiple users.**

Suggested solutions:

- Check that work is reaching the system under test. Ensure that some external device does not limit the amount of work reaching the system. Tivoli Performance Viewer helps determine the number of requests in the system.
- A thread dump might reveal a bottleneck at a synchronized method or a large number of threads waiting for a resource.
• Make sure that enough threads are available to process the work both in IBM HTTP Server, database, and the application servers. Conversely, too many threads can increase resource contention and reduce throughput.
• Monitor garbage collections with Tivoli Performance Viewer or the verbosegc option of your Java virtual machine. Excessive garbage collection can limit throughput.

**Methodology Best Practices**

If you need assistance, IBM Software Services for WebSphere (ISSW) provides professional consultants to help: [http://www.ibm.com/developerworks/middleware/services/](http://www.ibm.com/developerworks/middleware/services/)

1. Methodically capture data and logs for each test and record results in a spreadsheet. In general, it is best to change one variable at a time. Example test matrix:

<table>
<thead>
<tr>
<th>Test #</th>
<th>Start Time</th>
<th>Ramped Up</th>
<th>End Time</th>
<th>Concurrent Users</th>
<th>Average Throughput (Responses per Second)</th>
<th>Average Response Time (ms)</th>
<th>Average WAS CPU%</th>
<th>Average Database CPU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/1/201 14:00 GMT</td>
<td>1/1/201 14:30 GMT</td>
<td>1/1/201 16:00 GMT</td>
<td>10</td>
<td>50</td>
<td>100</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

2. Use a flow chart that everyone agrees to. Otherwise, alpha personalities or haphazard and random testing are likely to prevail, and these are less likely to succeed. The following is just an example.
Depth first means first "fill in" application server JVMs within a node before scaling across multiple nodes. The following are example hypotheses that are covered in more detail in each product chapter. They are summarized here just for illustration of hypotheses:

- CPU is low, so we can increase threads.
- CPU is low, so there is lock contention (gather monitor contention data through a sampling profiler such as IBM WAIT or IBM Java Health Center).
- CPU is high, so we can decrease threads or investigate possible code issues (gather profiling data through a sampling profiler such as IBM WAIT or IBM Java Health Center).
- Garbage collection overhead is high, so we can tune it.
- Connection pool wait times are high, so we can increase the size of the connection pool (if the total number of connections do not exceed the limits in the database).
- Database response times are high (also identified in thread dumps with many threads stuck in SQL calls), so we can investigate the database.

3. Deeply understand the logical, physical, and network layout of the systems. Create a rough diagram of the relevant components and important details. For example, how are the various systems connected and do they share any resources (potential bottlenecks) such as networks, buses, etc? Are the operating systems virtualized? It’s also useful to understand the processor layout and in particular, the L2/L3 cache (and NUMA) layouts as you may want to "carve out" processor sets along these boundaries.

4. Most, if not all, benchmarks have a target maximum concurrent user count. This is usually the best place to start when tuning the various queue sizes, thread pools, etc.
5. Averages should be used instead of spot observations. For important statistics such as throughput, getting standard deviations would be ideal.

6. Each test should have a sufficient "ramp up" period before data collection starts. Applications may take time to cache certain content and the Java JIT will take time to optimally compile hot methods.

7. Monitor all parts of the end-to-end system.

8. Consider starting with an extremely simplified application to ensure that the desired throughput can be achieved. Incrementally exercise each component: for example, a Hello World servlet, followed by a servlet that does a simple select from a database, etc. This lets you confirm that end-to-end "basics" work, including the load testing apparatus.

9. Run a saturation test where everything is pushed to the maximum (may be difficult due to lack of test data or test machines). Make sure things don't crash or break.

**Is changing one variable at a time always correct?**

It's common wisdom that one should always change one variable at a time when investigating problems, performance testing, etc. The idea is that if you change more than one variable at a time, and the problem goes away, then you don't know which one solved it. For example, let's say one changes the garbage collection policy, maximum heap size, and some of the application code, and performance improves, then one doesn't know what helped.

The premise underlying this wisdom is that all variables are independent, which is sometimes (maybe usually, to different degrees) not the case. In the example above, the garbage collection policy and maximum heap size are intimately related. For example, if you change the GC policy to gencon but don't increase the maximum heap size, it may not be a fair comparison to a non-gencon GC policy, because the design of gencon means that some proportion of the heap is no longer available relative to non-gencon policies (due to the survivor space in the nursery, based on the tilt ratio). See [http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html#migrating](http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html#migrating)

What's even more complicated is that it's often difficult to reason about variable independence. For example, most variables have indirect effects on processor usage or other shared resources, and these can have subtle effects on other variables. The best example is removing a bottleneck at one tier overloads another tier and indirectly affects the first tier (or exercises a new, worse bottleneck).

So what should one do? To start, accept that changing one variable at a time is not always correct; however, it's often a good starting point. Unless there's a reason to believe that changing multiple, dependent variables makes sense (for example, comparing gencon to non-gencon GC policies), then it's fair to assume initially that, even if variables may not be truly independent, the impact of one variable commonly drowns out other variables.

Just remember that ideally you would test all combinations of the variables. Unfortunately, as the number of variables increases, the number of tests increases exponentially. Specifically, for N variables, there are \((2^N - 1)\) combinations. For example, for two variables A and B, you would test A by itself, B by itself, and then A and B together \((2^2 - 1 = 3)\). However, by just adding two more variables to make the total four variables, it goes up to 15 different tests.

There are three reasons to consider this question:

First, it's an oversimplification to think that one should always change one variable at a time, and it's
important to keep in the back of one's head that if changing one variable at a time doesn't work, then changing multiple variables at a time might (of course, they might also just be wrong or inconsequential variables).

Second, particularly for performance testing, even if changing a single variable improves performance, it's possible that changing some combination of variables will improve performance even more. Which is to say that changing a single variable at a time is non-exhaustive.

Finally, it's not unreasonable to try the alternative, scattershot approach first of changing all relevant variables at the same time, and if there are benefits, removing variables until the key ones are isolated. This is more risky because there could be one variable that makes an improvement and another that cancels that improvement out, and one may conclude too much from this test. However, one can also get lucky by observing some interesting behavior from the results and then deducing what the important variable(s) are. This is sometimes helpful when one doesn't have much time and is feeling lucky (or has some gut feelings to support this approach).

So what's the answer to the question, "Is changing one variable at a time always correct?"

No, it's not always correct. Moreover, it's not even optimal, because it's non-exhaustive. But it usually works.

**Keep a Playbook**

When a naval ship declares "battle stations" there is an operations manual that every sailor on the ship is familiar with, knows where they need to go and what they need to do. Much like any navy when a problem occurs that negatively affects the runtime environment it is helpful for everyone to know where they need to be and who does what.

Each issue that occurs is an educational experience. Effective organizations have someone on the team taking notes. This way when history repeats itself the team can react more efficiently. Even if a problem does not reappear the recorded knowledge will live on. Organizations are not static. People move on to new projects and roles. The newly incoming operations team members can inherit the documentation to see how previous problems were solved.

For each problem we want to keep a record of the following points:

1. Symptom(s) of the problem - brief title
   • Who reported the problem?  
   • What exactly is the problem?  
3. Summary of all the people that were involved in troubleshooting and what was their role? The role is important because it will help the new team understand what roles need to exist.  
4. Details of  
   • What data was collected?  
   • Who looked at the data?  
   • The result of their analysis  
   • What recommendations were made  
   • Did the recommendations work (i.e. fix the problem)?
Statistics

Basic statistical definitions:

- **Average/Mean (μ):** An average is most commonly an arithmetic mean of a set of values, calculated as the sum of a set of values, divided by the count of values: \( \mu = \frac{\sum_{i=1}^{N} x_i}{N} \). For example, to calculate the average of the set of values (10, 3, 3, 1, 99), sum the values (116), and divide by the count, 5 (\( \mu = 23.2 \)).

- **Median:** A median is the middle value of a sorted set of values. For example, to calculate the median of the set of values (10, 3, 3, 1, 99), sort the values (1, 3, 3, 10, 99), and take the midpoint value (3). If the count of values is even, then the median is the average of the middle two values.

- **Mode:** A mode is the value that occurs most frequently. For example, to calculate the mode of the set of values (10, 3, 3, 1, 99), find the value that occurs the most times (3). If multiple values share this property, then the set is multi-modal.

- **Standard Deviation (σ):** A standard deviation is a measure of how far a set of values are spread out relative to the mean, with a standard deviation of zero meaning all values are equal, and more generally, the smaller the standard deviation, the more the values are closer to the mean. If the set of values is the entire population of values, then the standard deviation is calculated as the square root of the average of the squared differences from the mean: 
  \[ \sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}} \]. If the set of values is a sample from the entire population, then the sample standard deviation uses the division \((N - 1)\) instead of \(N\).

- **Confidence Interval:** A confidence interval describes the range of values in which the true mean has a high likelihood of falling (usually 95%), assuming that the original random variable is normally distributed, and the samples are independent. If two confidence intervals do not overlap, then it can be concluded that there is a difference at the specified level of confidence in performance between two sets of tests.

- **Relative change:** The ratio of the difference of a new quantity (B) minus an old quantity (A) to the old quantity: \( \frac{B - A}{A} \). Multiply by 100 to get the percent change. If A is a "reference value" (e.g. theoretical, expected, optimal, starting, etc.), then relative/percent change is relative/percent difference.

Small sample sizes (N) and large variability (σ) decrease the likelihood of correct interpretations of test results.

Here is R code that shows each of these calculations (the R project is covered under the Major Tools chapter):

```r
> values=c(10, 3, 3, 1, 99)
> mean(values)
[1] 23.2
> median(values)
[1] 3
> summary(values) # A quick way to do the above
   Min. 1st Qu. Median 3rd Qu. Max. 
   1.0 3.0  3.0 23.2 10.0 99.0
> mode = function(x) { ux = unique(x); ux[which.max(tabulate(match(x, ux)))] }
> mode(values)
[1] 3
```
> sd(values) # Sample Standard Deviation
[1] 42.51118
> error = qt(0.975, df=length(values)-1)*(sd(values)/sqrt(length(values)))
> ci = c(mean(values) - error, mean(values) + error)
> ci # Confidence Interval at 95%
[1] -29.5846 75.9846

Amdahl's Law

Amdahl's Law states that the maximum expected improvement to a system when adding more parallelism (e.g. more CPUs of the same speed) is limited by the time needed for the serialized portions of work. The general formula is not practically calculable for common workloads because they usually include independent units of work; however, the result of Amdahl's Law for common workloads is that there are fundamental limits of parallelization for system improvement as a function of serialized execution times.

In general, because no current computer system avoids serialization completely, Amdahl's Law shows that, all other things equal, the throughput curve of a computer system will approach an asymptote (which is limited by the bottlenecks of the system) as number of concurrent users increases (https://arxiv.org/pdf/cs/0404043v1.pdf):

![Canonical throughput characteristic](image)

Fig. 1. Canonical throughput characteristic.

Relatedly, response times follow a hockey stick pattern once saturation occurs:
Fig. 3 shows the canonical system response time characteristic $R$ (the dark curve). This shape is often referred to as the response hockey stick. It is the kind of curve that would be generated by taking time-averaged delay measurements in steady state at successive client loads. The dashed lines in Fig. 3 also represent bounds on the response time characteristic. The horizontal dashed line is the floor of the achievable response time $R_{\text{min}}$. It represents the shortest possible time for a request to get through the system in the absence of any contention. The sloping dashed line shows the worst case response time once saturation has set in. (https://arxiv.org/pdf/cs/0404043v1.pdf)

Queuing Theory

Queuing theory (https://en.wikipedia.org/wiki/Queueing_theory) is a branch of mathematics that may help model, analyze, and predict the behavior of queues when requests (e.g. HTTP requests) flow through a set of servers (e.g. application threads) or a network of queues. The models are approximations with various assumptions that may or may not be applicable in real world situations. There are a few key things to remember:

- A server is the thing that actually processes a request (e.g. an application thread).
- A queue is a buffer in front of the servers that holds requests until a server is ready to process them (e.g. a socket backlog, or a thread waiting for a connection from a pool).
• The arrival rate (λ) is the rate at which requests enter a queue. It is often assumed to have the characteristics of a random/stochastic/Markovian distribution such as the Poisson distribution (https://en.wikipedia.org/wiki/Poisson_distribution).
• The service time (µ) is the average response time of servers at a queue. Similar to the arrival rate, it is often assumed to have the characteristics of a Markovian distribution such as the Exponential distribution (https://en.wikipedia.org/wiki/Exponential_distribution).
• Queues are described using Kendall's notation (https://en.wikipedia.org/wiki/Kendall%27s_notation): A/S/c
  • A is the distribution of arrivals, which is normally M for Markovian (e.g. Poisson),
  • S is the distribution of service times, which is normally M for Markovian (e.g. Exponential),
  • c is the number of concurrent servers (e.g. threads).
• Therefore, the most common type of a queue we will deal with is an M/M/c queue.

For example, we will model a typical three tier architecture with a web server (e.g. IHS), application server (e.g. WAS), and a database:

<table>
<thead>
<tr>
<th>Open Queuing Network Model of 3-Tier Web Service</th>
</tr>
</thead>
</table>

This is a queuing network of three multi-server queues in series. Steady state analysis can be done by analyzing each tier independently as a multi-server M/M/c queue. This is so because it was proved that in a network where multi-server queues are arranged in series, the steady state departure processes of each queue are the same as the arrival process of the next queue. That is, if the arrival process in the first multi-server queue is Poisson with parameter λ then the steady state departure process of the same
Assumptions:

- The arrival process is Poisson with rate $\lambda$. That is, the inter-arrival time $T_1$ between arrivals of two successive requests (customers) is exponentially distributed with parameter $\lambda$. This means:
  \[
  \Pr \{ T_1 \leq t \} = 1 - e^{-\lambda t}, \quad t > 0
  \]
- The service rate of each server is exponentially distributed with parameter $\mu$, that is the distribution of the service time is:
  \[
  T = \Pr \{ T \leq t \} = 1 - e^{-\mu t}, \quad t > 0
  \]

1. Stability Condition: The arrival rate has to be less than the service rate of $m$ servers together. That is:

\[
\lambda < m\mu \quad \text{or} \quad \rho = \frac{\lambda}{m\mu} < 1
\]

2. State Occupancy Probability:

$\pi_i = \text{Probability that there are } i \text{ customers (requests) in the system (at service)}$

$\pi_{m+k} = \text{Probability that there are } m+k \text{ customers (requests) in the system (} m \text{ at service and } k \text{ waiting in the queue)}$

\[
\pi_i = \frac{(m\rho)^i}{i!} \pi_0, \text{ where } i = 0, 1, 2, \ldots, m
\]

\[
\pi_{m+k} = \rho^k \pi_m, \text{ where } k = 0, \ldots
\]

\[
\pi_0 = \frac{1}{\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!} \left( \frac{1}{1-\rho} \right)}
\]

3. Probability that a Customer (Request) has to Wait:
4. Expected number of Busy Servers:

\[ B = \frac{\lambda}{\mu} \]

5. Expected number of Waiting Requests:

\[ L_q = p_m \frac{\rho}{(1-\rho)^2} = \frac{\left(\frac{(m\rho)^m}{m!}\right)\left(\frac{\rho}{(1-\rho)^2}\right)}{\sum_{k=0}^{m-1} \frac{(m\rho)^k}{k!} + \frac{(m\rho)^m}{m!}\left(\frac{1}{1-\rho}\right)} \]

6. Expected Waiting Time in the Queue:

\[ W_q = \frac{L_q}{\lambda} \]

7. Expected Waiting Time in the Queue:

\[ L = L_q + B \]

8. Expected Waiting Time in the System:

\[ W = \frac{L}{\lambda} = W_q + \frac{1}{\mu} \]

To obtain performance measures of the Web Server, Application Server and Database Server, we replace \( m \) in the above given formulae by \( N^{WS} \), \( N^{AS} \) and \( N^{DS} \), respectively and replace \( \mu \) by \( 1/T^{WS} \), \( 1/T^{AS} \) and \( 1/T^{DS} \), respectively. As an example, the performance measures for the Web Server are given below. The performance measures for App Server and the DB Server can be obtained in the same way.
1W. Stability Condition for Web Server Queue:

\[ \lambda < N^{WS} \frac{1}{T^{WS}} \text{ or } \rho = \frac{\lambda T^{WS}}{N^{WS}} < 1 \]

2W. Web Server State Occupancy Probability:

\[ p_i = \text{Probability that there are } i \text{ customers (requests) in the system (at service)} \]

\[ p_{N^{WS}+k} = \text{Probability that there are } N^{WS}+k \text{ customers (requests) in the system (} N^{WS} \text{ at service and } k \text{ waiting in the queue)} \]

\[ p_i = \frac{(N^{WS}\rho)^i}{i!} \rho_0, \text{where } i = 0, 1, 2, \ldots, N^{WS} \]

\[ p_{N^{WS}+k} = \rho^k p_{N^{WS}}, \text{where } k = 0, \ldots \]

\[ p_0 = \frac{1}{\sum_{k=0}^{m-1} \frac{(N^{WS}\rho)^k}{k!} + \frac{(N^{WS}\rho)^m}{N^{WS}!} \left( \frac{1}{1-\rho} \right)} \]

3W. Probability that a Customer (Request) has to Wait at the Web Server:

\[ \frac{p_{N^{WS}}}{1-\rho} = \frac{\left( \frac{(N^{WS}\rho)^{N^{WS}}}{N^{WS}!} \right) \left( \frac{1}{1-\rho} \right)}{\sum_{k=0}^{N^{WS}-1} \frac{(N^{WS}\rho)^k}{k!} + \frac{(N^{WS}\rho)^{N^{WS}}}{N^{WS}!} \left( \frac{1}{1-\rho} \right)} \]

4W. Expected number of Busy Web Servers:
Little's Law

Little's Law states that the long-term average number of requests in a stable system \(L\) is equal to the long-term average effective arrival rate, \(\lambda\), multiplied by the (Palm-)average time a customer spends in the system, \(W\); or expressed algebraically: \(L = \lambda W\) (https://en.wikipedia.org/wiki/Little%27s_law).

Use Cases to Test Cases

Applications are typically designed with specific end user scenarios documented as use cases (for example, see the book Writing Effective Use Cases by Alistair Cockburn). Use cases drive the test cases that are created for load testing.
100% vs 80/20 rule?

A common perception in IT is performance testing can be accommodated by what is know as the 80/20 rule: We will test what 80% of actions the users do and ignore the 20% they do not as frequently. However, what is not addressed are the 20% that can induce a negative performance event causing serious performance degradation to the other 80%. Performance testing should always test 100% of the documented use cases.

The 80/20 rule also applies to how far you should tune. You can increase performance by disabling things such as performance metrics (PMI) and logging, but this may sacrifice serviceability and maintenance. Unless you're actually benchmarking for top speed, then we do not recommend applying such tuning.

Load Testing

Begin by choosing a benchmark, a standard set of operations to run. This benchmark exercises those application functions experiencing performance problems. Complex systems frequently need a warm-up period to cache objects, optimize code paths, and so on. System performance during the warm-up period is usually much slower than after the warm-up period. The benchmark must be able to generate work that warms up the system prior to recording the measurements that are used for performance analysis. Depending on the system complexity, a warm-up period can range from a few thousand transactions to longer than 30 minutes.

Another key requirement is that the benchmark must be able to produce repeatable results. If the results vary more than a few percent from one run to another, consider the possibility that the initial state of the system might not be the same for each run, or the measurements are made during the warm-up period, or that the system is running additional workloads.

Several tools facilitate benchmark development. The tools range from tools that simply invoke a URL to script-based products that can interact with dynamic data generated by the application. IBM® Rational® has tools that can generate complex interactions with the system under test and simulate thousands of users. Producing a useful benchmark requires effort and needs to be part of the development process. Do not wait until an application goes into production to determine how to measure performance.

The benchmark records throughput and response time results in a form to allow graphing and other analysis techniques.


Reset as many variables possible on each test. This is most important for tests involving databases which tend to accumulate data and can negatively impact performance. If possible, data should be truncated & reloaded on each test.

Determine the level of performance that will be considered acceptable to the customer at the outset of the engagement. Ensure that the objective is clear, measurable, and achievable.
Examples of this are:

- To show that the proposed WESB application can mediate complex messages 10 KB in size at a rate of 1,000 per second or more, on an 8 processor core pSeries Power6 system.
- To show that a WPS workflow application can handle 600 simultaneous users, where each human user’s average think time is 5 minutes and the response time is under 5 seconds, on an 8 processor core pSeries Power6 system.

As a counter-example, here is a poorly defined objective: "WPS performance must not be the bottleneck in the overall throughput of the solution, and must be able to handle peak load." Problems with this objective include: how do you prove that WPS is not the bottleneck? What is the definition of peak load? What are the attributes of the solution? What hardware will be used?

https://w3quickplace.lotus.com/QuickPlace/wasperf/PageLibrary852569AF00670F15.nsf/$defaultview/1CCEB50DD9A9C561852576030042A65C/$File/WebSphere%20BPM%206.2%20How%20To%20Win%20Performance%20POCs.pdf?OpenElement

**Stress Testing Tool**

There are various commercial products such as Rational Performance Tester (http://www-03.ibm.com/software/products/en/performance/). If such a tool is not available, there are various open source alternatives such as Apache Bench, Apache JMeter, Siege, and OpenSTA. The Apache JMeter tool is covered in more detail in the [Major Tools chapter](#) and it is a generally recommended tool.

**Apache Bench**

Apache Bench is a binary distributed in the "bin" folder of the httpd package (and therefore with IBM HTTP Server as well). It can do very simple benchmarking of a single URL, specifying the total number of requests (-n) and the concurrency at which to send the requests (-c). The most commonly useful parts of the output are highlighted in **bold** below:

```bash
$ ./ab -n 100 -c 5 http://ibm.com/
This is ApacheBench, Version 2.0.40-dev <$Revision: 30701 $> apache-2.0
Copyright (c) 1996 Adam Twiss, Zeus Technology Ltd
Copyright (c) 1998-2002 The Apache Software Foundation

Benchmarking ibm.com (be patient).....done

Server Software:        Apache/2.0.40-dev
Server Hostname:        ibm.com
Server Port:            80

Document Path:          /
Document Length:        227 bytes

Concurrency Level:       5
Time taken for tests:   2.402058 seconds
Complete requests:      100
```
Failed requests: 0
Write errors: 0
Non-2xx responses: 100
Total transferred: 49900 bytes
HTML transferred: 22700 bytes
Requests per second: 41.63 [#/sec] (mean)
Time per request: 120.103 [ms] (mean)
Time per request: 24.021 [ms] (mean, across all concurrent requests)
Transfer rate: 19.98 [Kbytes/sec] received

Connection Times (ms)

<table>
<thead>
<tr>
<th></th>
<th>min</th>
<th>mean[+/-sd]</th>
<th>median</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connect</td>
<td>44</td>
<td>56</td>
<td>8.1</td>
<td>55</td>
</tr>
<tr>
<td>Processing</td>
<td>51</td>
<td>61</td>
<td>6.9</td>
<td>60</td>
</tr>
<tr>
<td>Waiting</td>
<td>51</td>
<td>60</td>
<td>6.8</td>
<td>59</td>
</tr>
<tr>
<td>Total</td>
<td>97</td>
<td>117</td>
<td>12.1</td>
<td>115</td>
</tr>
</tbody>
</table>

Percentage of the requests served within a certain time (ms)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>115</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66%</td>
<td>124</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td>126</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80%</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90%</td>
<td>132</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td>141</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>98%</td>
<td>149</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99%</td>
<td>149</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td>149</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Common Benchmarks

DayTrader

DayTrader is a benchmark application built around the paradigm of an online stock trading system... The application allows users to log in, view their portfolio, look up stock quotes, and buy or sell stock shares... the real-world workload provided by DayTrader can be used to measure and compare the performance of Java Platform, Enterprise Edition (Java EE) application servers offered by a variety of vendors. ([http://geronimo.apache.org/GMOxDOC20/daytrader.html](http://geronimo.apache.org/GMOxDOC20/daytrader.html))

In order for you to evaluate a few common Java EE persistence and transaction management patterns, DayTrader provides three different implementations of the business services ([http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html](http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html)):

- Direct (Servlet to JDBC): Create, read, update, and delete (CRUD) operations are performed directly against the database using custom JDBC code. Database connections, commits, and rollbacks are managed manually in the code.
- Session Direct (Servlet to Stateless SessionBean to JDBC): CRUD operations are performed directly against the database using custom JDBC code. Database connections are managed manually in the code. Database commits and rollbacks are managed automatically by the stateless session bean.
- EJB (Servlet to StatelessSessionBean to EntityBean): The EJB container assumes responsibility
Think Times

Think time is defined to be the amount of time a user spends between requests. The amount of time a user spends on the page depends on how complex the page is and how long it takes for the user to find the next action to take. The less complex the page the less time it will take for the user to take the next action. However, no two users are the same so there is some variability between users. Therefore think time is generally defined as a time range, such as 4-15 seconds, and the load test tool will attempt to drive load within the parameters of think time. Testing that incorporates think time is attempting to simulate live production work load in order to attempt to tune the application for optimal performance.

There is also a "stress" test where think time is turned down to zero. Stress testing is typically used to simulate a negative production event where some of the application servers may have gone off line and are putting undue load on those remaining. Stress testing helps to understand how the application will perform during such a negative event in order to help the operations team understand what to expect. Stress testing also typically breaks the application in ways not encountered with normal think time testing. Therefore, stress testing is a great way to both:

- Break the application and have an attempt to fix it before being placed in production, and
- Providing the operations staff with information about what production will look like during a negative event.

Operating Systems

Additionally, see the chapter for your particular operating system:

- Linux
- AIX
- z/OS
- IBM i
- Windows
- Solaris
- HP-UX
- macOS

Central Processing Unit (CPU)

A processor is an integrated circuit (also known as a socket or die) with one or more central processing unit (CPU) cores. A CPU core executes program instructions such as arithmetic, logic, and input/output operations. CPU utilization is the percentage of time that programs or the operating system execute as opposed to idle time. A CPU core may support simultaneous multithreading (also known as hardware threads or hyperthreads) which appears to the operating system as additional logical CPU cores. Be aware that simple CPU utilization numbers may be unintuitive in the context of advanced processor features:

The current implementation of [CPU utilization]... shows the portion of time slots that the CPU scheduler in the OS could assign to execution of running programs or the OS itself;
the rest of the time is idle... The advances in computer architecture made this algorithm an unreliable metric because of introduction of multi core and multi CPU systems, multi-level caches, non-uniform memory, simultaneous multithreading (SMT), pipelining, out-of-order execution, etc.

A prominent example is the non-linear CPU utilization on processors with Intel® Hyper-Threading Technology (Intel® HT Technology). Intel® HT technology is a great performance feature that can boost performance by up to 30%. However, HT-unaware end users get easily confused by the reported CPU utilization: Consider an application that runs a single thread on each physical core. Then, the reported CPU utilization is 50% even though the application can use up to 70%-100% of the execution units. (https://software.intel.com/en-us/articles/intel-performance-counter-monitor)

Use care when partitioning [CPU cores]... it’s important to recognize that [CPU core] partitioning doesn’t create more resources, it simply enables you to divide and allocate the [CPU core] capacity... At the end of the day, there still needs to be adequate underlying physical CPU capacity to meet response time and throughput requirements when partitioning [CPU cores]. Otherwise, poor performance will result.


It is not necessarily problematic for a machine to have many more program threads than processor cores. This is common with Java and WAS processes that come with many different threads and thread pools by default that may not be used often. Even if the main application thread pool (or the sum of these across processes) exceeds the number of processor cores, this is only concerning if the average unit of work uses the processor heavily. For example, if threads are mostly I/O bound to a database, then it may not be a problem to have many more threads than cores. There are potential costs to threads even if they are usually sleeping, but these may be acceptable. The danger is when the concurrent workload on available threads exceeds processor capacity. There are cases where thread pools are excessively large but there has not been a condition where they have all filled up (whether due to workload or a front-end bottleneck). It is very important that stress tests saturate all commonly used thread pools to observe worst case behavior.

Depending on the environment, number of processes, redundancy, continuous availability and/or high availability requirements, the threshold for %CPU utilization varies. For high availability and continuous availability environments, the threshold can be as low as 50% CPU utilization. For non-critical applications, the threshold could be as high as 95%. Analyze both the non-functional requirements and service level agreements of the application in order to determine appropriate thresholds to indicate a potential health issue.

It is common for some modern processors (including server class) and operating systems to enable processor scaling by default. The purpose of processor scaling is primarily to reduce power consumption. Processor scaling dynamically changes the frequency of the processor(s), and therefore may impact performance. In general, processor scaling should not kick in during periods of high use; however, it does introduce an extra performance variable. Weigh the energy saving benefits versus disabling processor scaling and simply running the processors at maximum speed at all times (usually done in the BIOS).
Test affinitizing processes to processor sets (operating system specific configuration). In general, affinitize within processor boundaries. Also, start each JVM with -XgcthreadsN (IBM Java) or -XX:ParallelGCTreads=N (Oracle/HotSpot Java) where N equals the number of processor core threads in the processor set.

It is sometimes worth understanding the physical architecture of the central processing units (CPUs). Clock speed and number of cores/hyperthreading are the most obviously important metrics, but CPU memory locality, bus speeds, and L2/L3 cache sizes are sometimes worth considering. One strategy for deciding on the number of JVMs is to create one JVM per processor chip (i.e. socket) and bind it to that chip.

It's common for operating systems to dedicate some subset of CPU cores for interrupt processing and this may distort other workloads running on those cores.

Different types of CPU issues (Old Java Diagnostic Guide):

- Inefficient or looping code is running. A specific thread or a group of threads is taking all the CPU time.
- Points of contention or delay exist. CPU usage is spread across most threads, but overall CPU usage is low.
- A deadlock is present. No CPU is being used.

**How many CPUs per node?**

As a starting point, I plan on having at least one CPU [core] per application server JVM; that way I have likely minimized the number of times that a context switch will occur -- at least as far as using up a time slice is concerned (although, as mentioned, there are other factors that can result in a context switch). Unless you run all your servers at 100% CPU, more than likely there are CPU cycles available as application requests arrive at an application server, which in turn are translated into requests for operating system resources. Therefore, we can probably run more application servers than CPUs.

Arriving at the precise number that you can run in your environment, however, brings us back to it depends. This is because that number will in fact depend on the load, application, throughput, and response time requirements, and so on, and the only way to determine a precise number is to run tests in your environment.


**How many application processes per node?**

In general one should tune a single instance of an application server for throughput and performance, then incrementally add [processes] testing performance and throughput as each [process] is added. By proceeding in this manner one can determine what number of [processes] provide the optimal throughput and performance for their environment. In general once CPU utilization reaches 75% little, if any, improvement in throughput will be realized by adding additional [processes].
Random Access Memory (RAM), Physical Memory

Random access memory (RAM) is a high speed, ephemeral data storage circuit located near CPU cores. RAM is often referred to as physical memory to contrast it to virtual memory. Physical memory comprises the physical storage units which support memory usage in a computer (apart from CPU core memory registers), whereas virtual memory is a logical feature that an operating system provides for isolating and simplifying access to physical memory. Strictly speaking, physical memory and RAM are not synonymous because physical memory includes paging space, and paging space is not RAM.

Paging, Swapping

Paging space is a subset of physical memory, often disk storage or a solid state drive (SSD), which the operating system uses as a "spillover" when demands for physical memory exceed available RAM. Historically, swapping referred to paging in or out an entire process; however, many use paging and swapping interchangeably today, and both address page-sized units of memory (e.g. 4KB).

Overcommitting Memory

Overcommitting memory occurs when less RAM is available than the peak in-use memory demand. This is either done accidentally (undersizing) or consciously with the premise that it is unlikely that all required memory will be accessed at once. Overcommitting is dangerous because the process of paging in and out may be time consuming. RAM operates at over 10s of GB/s, whereas even the fastest SSDs operate at a maximum of a few GB/s (often the bottleneck is the interface to the SSD, e.g. SATA, etc.). Overcommitting memory is particularly dangerous with Java because some types of garbage collections will need to read most of the whole virtual address space for a process in a short period of time. When paging is very heavy, this is called memory thrashing, and usually this will result in a total performance degradation of the system of multiple magnitudes.

Sizing Paging Space

Some people recommend sizing the paging files to some multiple of RAM; however, this recommendation is a rule of thumb that may not be applicable to many workloads. Some people argue that paging is worse than crashing because a system can enter a zombie-like state and the effect can last hours before an administrator is alerted and investigates the issue. Investigation itself may be difficult because connecting to the system may be slow or impossible while it is thrashing. Therefore, some decide to dramatically reduce paging space (e.g. 10 MB) or remove the paging space completely which will force the operating system to crash processes that are using too much memory. This creates clear and immediate symptoms and allows the system to potentially restart the processes and recover. A tiny paging space is probably preferable to no paging space in case the operating system decides to do some benign paging. A tiny paging space can also be monitored as a symptom of problems.

Some workloads may benefit from a decently sized paging space. For example, infrequently used pages may be paged out to make room for filecache, etc.
"Although most do it, basing page file size as a function of RAM makes no sense because the more memory you have, the less likely you are to need to page data out." (Russinovich & Solomon)

**Non-uniform Memory Access (NUMA)**

Non-uniform Memory Access (NUMA) is a design in which RAM is partitioned so that subsets of RAM (called NUMA nodes) are "local" to certain processors. Consider affinitizing processes to particular NUMA nodes.

**32-bit vs 64-bit**

Whether 32-bit or 64-bit will be faster depends on the application, workload, physical hardware, and other variables. All else being equal, in general, 32-bit will be faster than 64-bit because 64-bit doubles the pointer size, therefore creating more memory pressure (lower CPU cache hits, TLB, etc.). However, all things are rarely equal. For example, 64-bit often provides more CPU registers than 32-bit (this is not always the case, such as Power), and in some cases, the benefits of more registers outweigh the memory pressure costs. There are other cases such as some mathematical operations where 64-bit will be faster due to instruction availability (and this may apply with some TLS usage, not just obscure mathematical applications). Java significantly reduces the impact of the larger 64-bit pointers within the Java heap by using compressed references. With all of that said, in general, the industry is moving towards 64-bit and the performance difference for most applications is in the 5% range.

**Large Page Support**

Several platforms support using memory pages that are larger than the default memory page size. Depending on the platform, large memory page sizes can range from 4 MB (Windows) to 16 MB (AIX) and up to 1GB versus the default page size of 4KB. Many applications (including Java-based applications) often benefit from large pages due to a reduction in CPU overhead associated with managing smaller numbers of large pages. See [http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html#sec4c](http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html#sec4c)


Some recent benchmarks on very modern hardware have found little benefit to large pages, although no negative consequences so they're still a best practice in most cases.

**Input/Output (I/O)**

**Disk**

Many problems are caused by exhausted disk space. It is critical that disk space is monitored and alerts are created when usage is very high.

Disk speed may be an important factor in some types of workloads. Some operating systems support mounting physical memory as disk partitions (sometimes called RAMdisks), allowing you to target certain disk operations that have recreatable contents to physical memory instead of slower disks.
Network Interface Cards (NICs) and Switches

Ensure that NICs and switches are configured to use their top speeds and full duplex mode. Sometimes this needs to be explicitly done, so you should not assume that this is the case by default. In fact, it has been observed that when the NIC is configured for auto-negotiate, sometimes the NIC and the switch can auto-negotiate very slow speeds and half duplex. This is why setting explicit values is recommended.

If the network components support Jumbo Frames, consider enabling it across the relevant parts of the network.

Check network performance between two hosts. For example, make a 1 GB file (various operating system commands like dd or mkfile). Then test the network throughput by copying it using FTP, SCP, etc.

Monitor ping latency between hosts, particularly any periodic large deviations.

It is common to have separate NICs for incoming traffic (e.g. HTTP requests) and for backend traffic (e.g. database). In some cases and particularly on some operating systems, this setup may perform worse than a single NIC (as long as it doesn't saturate) probably due to interrupts and L2/L3 cache utilization side-effects.

TCP/IP

TCP/IP is used for most network communications such as HTTP, so understanding and optimizing the operating system TCP/IP stack can have dramatic upstream effects on your application.

TCP/IP is normally used in a fully duplexed mode meaning that communication can occur asynchronously in both directions. In such a mode, a distinction between "client" and "server" is arbitrary and sometimes can confuse investigations (for example, if a web browser is uploading a large HTTP POST body, it is first the "server" and then becomes the "client" when accepting the response). You should always think of a set of two sender and receiver channels for each TCP connection.

TCP/IP is a connection oriented protocol, unlike UDP, and so it requires handshakes (sets of packets) to start and close connections. The establishing handshake starts with a SYN packet from sender IP address A on an ephemeral local port X to receiver IP address B on a port Y (every TCP connection is uniquely identified by this 4-tuple). If the connection is accepted by B, then B sends back an acknowledgment (ACK) packet as well as its own SYN packet to establish the fully duplexed connection (SYN/ACK). Finally, A sends a final ACK packet to acknowledge the established connection. This handshake is commonly referred to as SYN, SYN/ACK, ACK.

A TCP/IPv4 packet has a 40 byte header (20 for TCP and 20 for IPv4).

Network performance debugging (often euphemistically called "TCP tuning") is extremely difficult because nearly all flaws have exactly the same symptom: reduced performance. For example, insufficient TCP buffer space is indistinguishable from excess packet loss (silently repaired by TCP retransmissions) because both flaws just slow the application, without any specific identifying symptoms.

The amount of data that can be in transit in the network, termed "Bandwidth-Delay-Product," or BDP for short, is simply the product of the bottleneck link bandwidth and the
Round Trip Time (RTT).

https://www.psc.edu/index.php/networking/641-tcp-tune

In general, the maximum socket receive and send buffer sizes should be greater than the average BDP.

**Flow Control & Receive/Send Buffers**

TCP/IP flow control allows a sender to send more packets before receiving acknowledgments for previous packets. Flow control also tries to ensure that a sender does not send data faster than a receiver can handle. The receiver includes a "window size" in each acknowledgment packet which tells the sender how much buffer room the receiver has for future packets. If the window size is 0, the sender should stop sending packets until it receives a TCP Window Update packet or an internal retry timer fires. If the window size is non-zero, but it is too small, then the sender may spend unnecessary time waiting for acknowledgments. The window sizes are directly affected by the rate at which the application can produce and consume packets (for example, if CPU is 100% then a program may be very slow at producing and consuming packets) as well as operating system TCP sending and receiving buffer size limits. The buffers are chunks of memory allocated and managed by the operating system to support TCP/IP flow control. It is generally advisable to increase these buffer size limits as much as operating system configuration, physical memory and the network architecture can support.

The maximum throughput based on the receiver window is rwnd/RTT.

**CLOSE_WAIT**

If a socket is ESTABLISHED, and one side (let's call it side X) calls close, then X sends a FIN packet to the other side (let's call it side Y) and X enters the FIN_WAIT_1 state. At this point, X can no longer write bytes to Y; however, Y may still write bytes to X (each TCP socket has two pipes).

When Y receives the FIN, it sends an ACK back and Y enters the CLOSE_WAIT state. When X receives the ACK, it enters the FIN_WAIT_2 state. Y's CLOSE_WAIT state may be read as "Y is waiting for the application inside Y to call close on its write pipe to X." At this point, the socket could stay in this condition indefinitely with Y writing bytes to X. Although this is a valid TCP use case to have a half-opened socket, it is an uncommon use case, so sockets in CLOSE_WAIT state are more commonly simply waiting for Y to close its half of the socket. If the number of sockets in CLOSE_WAIT are high or increasing over time, this may be caused by a leak of the socket object in Y, lack of resources, missing or incorrect logic to close the socket, etc. If sockets in CLOSE_WAIT continuously increase, at some point the process may receive file descriptor exhaustion or other socket errors and the only resolution is to restart the process to clear the sockets.

When Y closes its half of the socket by sending a FIN to X, then Y enters LAST_ACK. When X responds with an ACK on the FIN, then the Y socket is completely closed, and X enters the TIME_WAIT state for a certain period of time.

The above is a normal close; however, it is also possible that RST packets are used to close sockets.

**TIME_WAIT**

TCP sockets pass through various states such as LISTENING, ESTABLISHED, CLOSED, etc. One
particularly misunderstood state is the TIME_WAIT state which can sometimes cause scalability issues. A full duplex close occurs when sender A sends a FIN packet to B to initiate an active close (A enters FIN_WAIT_1 state). When B receives the FIN, it enters CLOSE_WAIT state and responds with an ACK. When A receives the ACK, A enters FIN_WAIT_2 state. Strictly speaking, B does not have to immediately close its channel (if it wanted to continue sending packets to A); however, in most cases it will initiate its own close by sending a FIN packet to A (B now goes into LAST_ACK state). When A receives the FIN, it enters TIME_WAIT and sends an ACK to B. The reason for the TIME_WAIT state is that there is no way for A to know that B received the ACK. The TCP specification defines the maximum segment lifetime (MSL) to be 2 minutes (this is the maximum time a packet can wander the net and stay valid). The operating system should ideally wait 2 times MSL to ensure that a retransmitted packet for the FIN/ACK doesn't collide with a newly established socket on the same port (for instance, if the port had been immediately reused without a TIME_WAIT and if other conditions such as total amount transferred on the packet, sequence number wrap, and retransmissions occur).

This behavior can cause scalability issues:

Because of TIME-WAIT state, a client program should choose a new local port number (i.e., a different connection) for each successive transaction. However, the TCP port field of 16 bits (less the "well-known" port space) provides only 64512 available user ports. This limits the total rate of transactions between any pair of hosts to a maximum of \(\frac{64512}{240} = 268\) per second. (https://tools.ietf.org/pdf/rfc1379.pdf)

Most operating systems do not use 4 minutes as the default TIME_WAIT duration because of the low probability of the wandering packet problem and other mitigating factors. Nevertheless, if you observe socket failures accompanied with large numbers of sockets in TIME_WAIT state, then you should reduce the TIME_WAIT duration further. Conversely, if you observe very strange behavior when new sockets are created that can't be otherwise explained, you should use 4 minutes as a test to ensure this is not a problem.

Finally, it's worth noting that some connections will not follow the FIN/ACK, FIN/ACK procedure, but may instead use FIN, FIN/ACK, ACK, or even just a RST packet (abortive close).

**Nagle's Algorithm (RFC 896, TCP_NODELAY)**

There is a special problem associated with small packets. When TCP is used for the transmission of single-character messages originating at a keyboard, the typical result is that 41 byte packets (one byte of data, 40 bytes of header) are transmitted for each byte of useful data. This 4000% overhead is annoying but tolerable on lightly loaded networks. On heavily loaded networks, however, the congestion resulting from this overhead can result in lost datagrams and retransmissions, as well as excessive propagation time caused by congestion in switching nodes and gateways.

The solution is to inhibit the sending of new TCP segments when new outgoing data arrives from the user if any previously transmitted data on the connection remains unacknowledged. (https://tools.ietf.org/html/rfc896)

In practice, enabling Nagle's algorithm (which is usually enabled by default) means that TCP will not send a new packet if another previous sent packet is still unacknowledged, unless it has "enough" coalesced data for a larger packet.
The native C setsockopt option to disable Nagle's algorithm is TCP_NODELAY:
http://pubs.opengroup.org/onlinepubs/009696699/basedefs/netinet/tcp.h.html

This option can usually be set globally at an operating system level.

This option is also exposed in Java's StandardSocketOptions to allow for setting a particular Java
socket option:
http://docs.oracle.com/javase/7/docs/api/java/net/StandardSocketOptions.html#TCP_NODELAY

In WebSphere Application Server, TCP_NODELAY is explicitly enabled by default for all WAS TCP
channel sockets.

**Delayed Acknowledgments (RFC 1122)**

TCP delayed acknowledgments was designed in the late 1980s in an environment of baud speed
modems. Delaying acknowledgments was a tactic used when communication over wide area networks
was really slow and the delaying would allow for piggy-backing acknowledgment packets to responses
within a window of a few hundred milliseconds. In modern networks, these added delays may cause
significant latencies in network communications.

Delayed acknowledgments is a completely separate function from Nagle's algorithm
(TCP_NODELAY). Both act to delay packets in certain situations. This can be very subtle; for
example, on AIX, the option for the former is tcp_nodelayack and the option for the latter is
tcp_nodelay.

Delayed ACKs defines the default behavior to delay acknowledgments up to 500 milliseconds (the
common default maximum is 200 milliseconds) from when a packet arrives (but no more than every
second segment) to reduce the number of ACK-only packets and ACK chatter because the ACKs may
piggy back on a response packet. It may be the case that disabling delayed ACKs, while increasing
network chatter and utilization (if an ACK only packet is sent where it used to piggy back a data
packet, then there will be an increase in total bytes sent because of the increase in the number of
packets and therefore TCP header bytes), may improve throughput and responsiveness. However, there
are also cases where delayed ACKs perform better. It is best to test the difference.

"A host that is receiving a stream of TCP data segments can increase efficiency in both the
Internet and the hosts by sending fewer than one ACK (acknowledgment) segment per data
segment received; this is known as a "delayed ACK" [TCP:5].

A TCP SHOULD implement a delayed ACK, but an ACK should not be excessively
delayed; in particular, the delay MUST be less than 0.5 seconds, and in a stream of full-
sized segments there SHOULD be an ACK for at least every second segment.

A delayed ACK gives the application an opportunity to update the window and perhaps to
send an immediate response. In particular, in the case of character-mode remote login, a
delayed ACK can reduce the number of segments sent by the server by a factor of 3 (ACK, window update, and echo character all combined in one segment).

In addition, on some large multi-user hosts, a delayed ACK can substantially reduce
protocol processing overhead by reducing the total number of packets to be processed
[TCP:5]. However, excessive delays on ACK’s can disturb the round-trip timing and packet

Delayed acknowledgments interacts poorly with Nagle's algorithm. For example, if A sent a packet to B, and B is waiting to send an acknowledgment to A until B has some data to send (Delayed Acknowledgments), and if A is waiting for the acknowledgment (Nagle's Algorithm), then a delay is introduced.

In Wireshark, you can look for the "Time delta from previous packet" entry for the ACK packet to determine the amount of time elapsed waiting for the ACK... Although delayed acknowledgment may adversely affect some applications..., it can improve performance for other network connections. (http://www-01.ibm.com/support/docview.wss?uid=swg21385899)

The pros of delayed acknowledgments are:

1. Reduce network chatter
2. Reduce potential network congestion
3. Reduce network interrupt processing (CPU)

The cons of delayed acknowledgments are:

1. Potentially reduce response times and throughput

In general, if two hosts are communicating on a LAN and there is sufficient additional network capacity and there is sufficient additional CPU interrupt processing capacity, then disabling delayed acknowledgments will tend to improve performance and throughput. However, this option is normally set at an operating system level, so if there are any sockets on the box that may go out to a WAN, then their performance and throughput may potentially be affected negatively. Even on a WAN, for 95% of modern internet connections, disabling delayed acknowledgments may prove beneficial. The most important thing to do is to test the change with real world traffic, and also include tests emulating users with very slow internet connections and very far distances to the customer data center (e.g. second long ping times) to understand any impact. The other potential impact of disabling delayed acknowledgments is that there will be more packets which just have the acknowledgment bit set but still have the TCP/IP header (40 or more bytes). This may cause higher network utilization and network CPU interrupts (and thus CPU usage). These two factors should be monitored before and after the change.

Selective Acknowledgments (SACK, RFC 2018)

"With the limited information available from cumulative acknowledgments, a TCP sender can only learn about a single lost packet per round trip time... [With a] Selective Acknowledgment (SACK) mechanism... the receiving TCP sends back SACK packets to the sender informing the sender of data that has been received. The sender can then retransmit only the missing data segments." (https://tools.ietf.org/html/rfc2018)

Listen Back Log

The listen back log is a limited size queue for each socket that holds pending sockets that have completed the SYN packet but that the process has not yet "accepted" (therefore they are not yet established). This back log is used as an overflow for sudden spikes of connections. If the listen back
log fills up any new connection attempts (SYN packets) will be rejected by the operating system (i.e. they'll fail). As with all queues, you should size them just big enough to handle a temporary but sudden spike, but not too large so that too much operating system resources are used which means that new connection attempts will fail fast when there is a backend problem. There is no science to this, but 511 is a common value.

**Keep-alive**

RFC 1122 defines a "keep-alive" mechanism to periodically send packets for idle connections to make sure they're still alive:

A "keep-alive" mechanism periodically probes the other end of a connection when the connection is otherwise idle, even when there is no data to be sent. The TCP specification does not include a keep-alive mechanism because it could:

1. cause perfectly good connections to break during transient Internet failures;
2. consume unnecessary bandwidth ("if no one is using the connection, who cares if it is still good?"); and
3. cost money for an Internet path that charges for packets.

Some TCP implementations, however, have included a keep-alive mechanism. To confirm that an idle connection is still active, these implementations send a probe segment designed to elicit a response from the peer TCP ([https://tools.ietf.org/html/rfc1122#page-101](https://tools.ietf.org/html/rfc1122#page-101)).

By default, keep-alive is disabled unless a socket specifies SO_KEEPALIVE when it is created. The default idle interval must be no less than 2 hours, but can be configured in the operating system.

**Domain Name Servers (DNS)**

Ensure that Domain Name Servers (DNS) are very responsive.

Consider setting high Time To Live (TTL) values for hosts that are unlikely to change.

If performance is very important or DNS response times have high variability, consider adding all major DNS lookups to each operating system's local DNS lookup file (e.g. /etc/hosts).

**Troubleshooting Network Issues**

One of the troubleshooting steps for slow response time issues is to sniff the network between all the network elements (e.g. HTTP server, application server, database, etc.). The most popular tool for sniffing and analyzing network data is Wireshark which is covered in the Major Tools chapter. Common errors are frequent retransmission requests (sometimes due to a bug in the switch or bad cabling).

**Antivirus / Security Products**

We have seen increasing cases of antivirus leading to significant performance problems. Companies are more likely to run quite intrusive antivirus even on critical, production machines. The antivirus settings are usually corporate-wide and may be inappropriate or insufficiently tuned for particular applications.
or workloads. In some cases, even when an antivirus administrator states that antivirus has been "disabled," there may still be kernel level modules that are still operational. In some cases, slowdowns are truly difficult to understand; for example, in one case a slowdown occurred because of a network issue communicating with the antivirus hub, but this occurred at a kernel-level driver in fully native code, so it was very difficult even to hypothesize that it was antivirus. You can use operating system level tools and sampling profilers to check for such cases, but they may not always be obvious. Keep a watch out for signs of antivirus and consider running a benchmark comparison with and without antivirus (completely disabled, perhaps even uninstalled).

Another class of products that are somewhat orthogonal are security products which provide integrity, security, and data scrubbing capabilities for sensitive data. For example, they will hook into the kernel so that any time a file is copied onto a USB key, a prompt will ask whether the information is confidential or not (and if so, perform encryption). This highlights the point that it is important to gather data on which kernel modules are active (e.g. using CPU during the time of the problem).

**Clocks**

To ensure that all clocks are synchronized on all nodes use something like the Network Time Protocol (NTP). This helps with correlating diagnostics and it's required for certain functions in products.

Consider setting one standardized time zone for all nodes, regardless of their physical location. Some consider it easier to standardize on the UTC/GMT/Zulu time zone.

**POSIX**

POSIX, or Portable Operating System Interface for Unix, is the public standard for Unix-like operating systems, including things like APIs, commands, utilities, threading libraries, etc. It is implemented in part or in full by: AIX, Linux, Solaris, z/OS USS, HP/UX, etc.

**Monitoring TCP Connections**

One simple and very useful indicator of process health and load is its TCP activity. Here is a script that takes a set of ports and summarizes how many TCP sockets are established, opening, and closing for each port: https://raw.githubusercontent.com/kgibm/problemetermination/master/scripts/netstat/portstats.sh. It has been tested on Linux and AIX. Example output:

```
$ portstats.sh 80 443
PORT  ESTABLISHED  OPENING  CLOSING
 80     3              0         0
 443    10            0          2
====================================
Total  13             0         2
```

**SSH Keys**

As environments continue to grow, automation becomes more important. On POSIX operating systems, SSH keys may be used to automate running commands, gathering logs, etc. A 30 minute investment to configure SSH keys will save countless hours and mistakes.
Step #1: Generate an "orchestrator" SSH key

1. Choose one of the machines that will be the orchestrator (or a Linux, Mac, or Windows cygwin machine)
2. Ensure the SSH key directory exists:
   $ cd ~/.ssh/
   If this directory does not exist:
   $ mkdir ~/.ssh && chmod 700 ~/.ssh && cd ~/.ssh/
3. Generate an SSH key:
   $ ssh-keygen -t rsa -b 4096 -f ~/.ssh/orchestrator

Step #2: Distribute "orchestrator" SSH key to all machines

If using Linux:

1. Run the following command for each machine:
   $ ssh-copy-id -i ~/.ssh/orchestrator user@host

For other POSIX operating systems

1. Log in to each machine as a user that has access to all logs (e.g. root):
   $ ssh user@host
2. Ensure the SSH key directory exists:
   $ cd ~/.ssh/
   If this directory does not exist:
   $ mkdir ~/.ssh && chmod 700 ~/.ssh && cd ~/.ssh/
3. If the file ~/.s
   $ touch ~/.s
4. Append the public key from ~/.s
   $ cat >> ~/.s
   Paste your clipboard and press ENTER
   Ctrl+D to save

Step #3: Now you are ready to automate things

Go back to the orchestrator machine and test the key:

1. Log into orchestrator machine and try to run a simple command on another machine:
   $ ssh -i ~/.ssh/orchestrator root@machine2 "hostname"
2. If your SSH key has a password, then you'll want to use ssh-agent so that it's cached for some time:
   $ ssh-add ~/.ssh/orchestrator
3. If this gives an error, try starting ssh-agent:
   $ ssh-agent
4. Now try the command again and it should give you a result without password:
   $ ssh -i ~/.ssh/orchestrator root@machine2 "hostname"

Now we can create scripts on the orchestrator machine to stop servers, clear logs, start servers, start mustgathers, gather logs, etc.
Example Scripts

In all the example scripts below, we basically iterate over a list of hosts and execute commands on all of those hosts. Remember that if the orchestrator machine is also one of these hosts, that it should be included in the list (it will be connecting to "itself"). You will need to modify these scripts to match what you need.

Example Script to Stop Servers

```bash
#!/bin/sh
USER=root
for i in ihs1hostname ihs2hostname; do
    ssh -i ~/.ssh/orchestrator $USER@$i "/opt/IBM/HTTPServer/bin/apachectl -k stop"
    ssh -i ~/.ssh/orchestrator $USER@$i "kill -INT `pgrep tcpdump`"
done
for i in wl1hostname wl2hostname; do
    ssh -i ~/.ssh/orchestrator $USER@$i "/opt/liberty/bin/server stop ProdSrv01"
    ssh -i ~/.ssh/orchestrator $USER@$i "kill -INT `pgrep tcpdump`"
done
```

Example Script to Clear Logs

```bash
#!/bin/sh
USER=root
for i in ihs1hostname ihs2hostname; do
    ssh -i ~/.ssh/orchestrator $USER@$i "rm -rf /opt/IBM/HTTPServer/logs/*"
    ssh -i ~/.ssh/orchestrator $USER@$i "rm -rf /opt/IBM/HTTPServer/Plugin/webserver1/logs/*"
    ssh -i ~/.ssh/orchestrator $USER@$i "nohup tcpdump -nn -v -i any -C 100 -W 10 -Z root -w /tmp/capture`date "+%Y%m%d_%H%M`".pcap &"
done
for i in wl1hostname wl2hostname; do
    ssh -i ~/.ssh/orchestrator $USER@$i "rm -rf /opt/liberty/usr/servers/*/logs/*"
    ssh -i ~/.ssh/orchestrator $USER@$i "nohup tcpdump -nn -v -i any -C 100 -W 10 -Z root -w /tmp/capture`date "+%Y%m%d_%H%M`".pcap &"
done
```

Example Script to Execute perfmustgather

```bash
#!/bin/sh
USER=root
for i in wl1hostname wl2hostname; do
    ssh -i ~/.ssh/orchestrator $USER@$i "/opt/perfMustGather.sh --outputDir /tmp/ --iters 6 `cat /opt/liberty/usr/servers/.pid/*.pid` &"
done
```

Example Script to Gather Logs

```bash
#!/bin/sh
USER=root
```
LOGS=`date`"%Y%m%d_%H%M``
mkdir $LOGS
for i in ihs1hostname ihs2hostname; do
  mkdir $LOGS/ihs/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/opt/IBM/HTTPServer/logs/* $LOGS/ihs/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/opt/IBM/HTTPServer/conf/httpd.conf $LOGS/ihs/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/opt/IBM/HTTPServer/plugings/config/*/plugin-cfg.xml $LOGS/ihs/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/opt/IBM/HTTPServer/Plugin/webserver1/logs/* $LOGS/ihs/$i/
done
for i in w1hostname w2hostname; do
  mkdir $LOGS/liberty/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/opt/liberty/usr/servers/*/logs/$LOGS/liberty/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/opt/liberty/usr/servers/*/server.xml $LOGS/liberty/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/tmp/capture*.pcap* $LOGS/liberty/$i/
  scp -r -i ~/.ssh/orchestrator $USER@$i:/tmp/mustgather_RESULTS.tar.gz $LOGS/liberty/$i/
done
  tar czvf $LOGS.tar.gz $LOGS

Ulimits

IBM often asks clients to increase process limits (ulimits) on POSIX operating systems such as AIX, Linux, Solaris, and HP-UX. The default process limits often cause truncated core dumps (also known as system dumps). Core dump truncation means that diagnostics produced by crashes and OutOfMemoryErrors are usually unusable and the problems need to be reproduced with increased ulimits.

Modern operating systems are based on the concept of multi-user, time-sharing systems. Operating systems use three key features to isolate users and processes from each other: user mode, virtual address spaces, and process/resource limits. Before these innovations, it was much easier for users and processes to affect each other, whether maliciously or not.

User mode forces processes to use system calls provided by the kernel instead of directly interacting with memory, devices, etc. This feature is ultimately enforced by the processor itself. Kernel code runs in a trusted, unrestricted mode, allowing it to do certain things that a user-mode process cannot do. A user-mode process can make a system call into the kernel to request such functions and this allows the kernel to enforce constraints and share limited resources.

Virtual address spaces allow each process to have its own memory space instead of managing and sharing direct memory accesses. The processor and kernel act in concert to allocate physical memory and paging space and translate virtual addresses to physical addresses. Again, this provides the ability to restrict which memory a process can access and in what way.

Process and resource limits are a subset of the restrictions enforced by the kernel and virtual address
spaces which can be configured on a per-process or per-user basis. The two key limits related to this
discussion are the core file size limit and the maximum file size limit. (On AIX, the fullcore and
pre430core system configuration settings are also important.) Process limits come in two flavors - soft
and hard - but for the purposes of this discussion, both should be set. Note that IBM Java will increase
its core file and maximum file size soft limits up to the allowed hard limits at runtime.

The core file ulimit and maximum file size ulimit are independent mechanisms to limit the maximum
size of a core dump, since a core dump is just a regular file. These default values are restricted for
many reasons, including:

1. Disk space: Core dumps are very big because they dump most of the virtual address space. Even
   a simple program may have a virtual address space of a few hundred megabytes, and the
   average Java program has a virtual address space of a few gigabytes. By default, a core dump
   will be created in the current working directory of the process. If core dumps are not managed
   properly, then they can consume available disk space which may cause other problems.
2. Security: Core dumps dump out most of the memory contents for that virtual address space for
   an instant in time. This may contain sensitive user information.
3. Historical inertia: Some of these defaults date back three or four decades.

**Is it safe to increase operating system process limits?**

If you increase core and maximum file size ulimits and you have insufficient disk space, then your
application may be affected (and the core dump itself may be truncated). For example, the default
working directory of a WAS process is the profile directory and a core file will be written to the default
working directory. This directory has many artifacts such as configuration, transaction information,
logs, and more, so if core files fill up this disk, then application updates and other functionality may
fail. Therefore, we recommend that a dedicated filesystem is created for system dumps. This filesystem
should be on a fast, local disk (or even a RAMdisk). Then you can either use -Xdump arguments to
change the path of system dumps, or change the "Working directory" under Application Servers >
SERVER > Process Definition.

You can change the default path of system dumps using the IBM Java -Xdump "defaults" option in a
generic JVM argument -Xdump:system:defaults:file=/somepath/core.%Y%m%d.%H%M%S.%pid.
%seq.dmp

The -Xdump parameter is only available on IBM Java, so it cannot be used on Solaris and HP

If you decide to change the working directory instead, be aware that this will also change where
javacores and other artifacts go, so you should document this change for your administrators' awareness.

We recommend unlimited core file and maximum file size ulimits because we cannot know ahead of
time how large the virtual address space will become. For example, if you have a native memory leak,
then the virtual address space can become very large before a problem occurs, and determining the
problem with a truncated core dump will be very difficult or impossible.

**What are the best practices for setting default operating system limits?**

You shouldn't wait until you have a problem. We recommend that proper ulimits are set ahead of time
for any WAS installation. In general, the easiest way is to update the relevant global ulimit
configuration file: http://www-01.ibm.com/support/docview.wss?uid=swg21469413. Then you will need to restart all WAS processes (and if you were logged in to a shell from which you restart such processes, you will need to log out and log back in).

For example, on Linux, if Java is run under the wasadm user name, you can add these lines to /etc/security/limits.conf:

```bash
wasadm soft core unlimited
wasadm hard core unlimited
wasadm soft fsize unlimited
wasadm hard fsize unlimited
```

Alternatively, you can set these ulimits in the shell that spawns the process before the Java command:

```bash
ulimit -c unlimited
ulimit -f unlimited
```

If you are starting servers from the Administrative Console or using wsadmin, then make sure that you set these ulimits in the node agent (startNode.sh) and restart it so that the server will inherit the node agent's ulimits. If you are manually starting servers using startServer.sh, then you must update startServer.sh itself or ensure ulimits are set in the shell where you launch startServer.sh.

**Other Performance considerations for unlimited ulimits**

While writing a core dump, the operating system will completely pause the process. After the core dump is finished, the process will resume where it left off. The time it takes to write the core dump is mostly proportional to the virtual address space size, available file cache in physical memory, and disk speed.

You can determine the virtual address space size in various ways (where VSZ is in KB):

```bash
AIX: ps -o pid,vsz -L PID
Linux: ps -o pid,vsz -p PID
Solaris: ps -o pid,vsz -p PID
HP-UX: UNIX95="" ps -o pid,vsz -p PID
```

The file cache is an area of physical memory (RAM) that is used as a write-behind or write-through cache for some virtual file system operations. If a file is created, written to, or read from, the operating system may try to perform some or all of these operations through physical memory and then flush any changes to disk. This dramatically improves performance of file I/O at the risk of losing file updates if a machine crashes before the data is flushed to disk. The best way to try to improve performance of writing system dumps is to ensure that physical memory and file cache have at least the amount of the virtual size of the process available. This means that the operating system will write the core to RAM, continue the process, and then asynchronously write it to disk. The filecache is very operating system specific and cannot be dedicated to particular file operations, so even if you increase your memory and file cache, other file I/O operations may fill it up (for example, if you are tracing heavily at the same time).

Finally, disk speed is a major factor in core dump writing performance for obvious reasons and you can increase performance by dedicating a faster disk for core dump processing.
How do you confirm that ulimits are set correctly?

The following IBM Java versions introduced a ulimit section in a javacore: Java 5 SR11 (WAS 6.1.0.29), Java 6 SR6 (WAS 7.0.0.7), Java 626 (WAS 8). You can take a javacore using "kill -3 PID" or other ways and search for this section:

```
1CUSERLIMITS User Limits (in bytes except for NOFILE and NPROC)
NULL -----------------------------------------------
NULL type soft limit hard limit
2CUSERLIMIT RLIMIT_CORE unlimited unlimited
2CUSERLIMIT RLIMIT_FSIZE unlimited unlimited
```

On some versions of Linux, you can simply cat /proc/PID/limits.

On Solaris, you can use the "plimit" command to print the current ulimits for a running process. On both Solaris and HP-UX, the hs_err_pid crash file will contain ulimits at the time of the crash.

Why use non-unlimited ulimits at all?

There is a valid philosophical question of whether to set every ulimit to unlimited for particular users such as those running WAS (note that some values do not accept "unlimited" such as maximum open files on Linux but instead take a specific maximum value). Changing ulimit values is not really a "tuning" exercise since these values are simply restrictions on what a process can do. If a machine is dedicated for particular purposes (WAS, IHS, etc.) then it makes sense to run these processes unrestricted and only restrict other processes. It is true that some of these resources directly or indirectly use kernel memory which is a shared, limited resource; however, a well-behaved kernel should kill any process that exhausts its resources and the resulting core dump should have obvious symptoms of the offending resource usage or leak. Until that point, it's not clear why potentially legitimate resource usage is constricted by these arbitrary default ulimits (or equally arbitrary ulimits you may find in other tuning documents).

Linux

Linux Recipe

1. **CPU core(s)** should not be consistently saturated. Use tools such as vmstat, top, atop, nmon, perf, SystemTap, etc.
2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk. Use tools such as free, vmstat, /proc/meminfo, top, atop, nmon, etc.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times. Use tools such as df, stat, iostat, netstat, ping, nfsiostat, etc.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic effects on performance. Tune TCP/IP socket buffers such as net.core.*mem* and net.ipv4.tcp_*mem* and monitor $(netstat -s) for TCP retransmissions.
5. Set **vm.swappiness=0** on systems running Java-based workloads which have light disk file I/O.
6. **Consider** disabling swap, setting vm.panic_on_oom=1, and configuring kernel vmcore dumps with process-level virtual address space information to avoid swap thrashing situations and reduce downtime, whilst analyzing post-mortem vmcores for excessive memory usage, leaks, or
undersizing.
7. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis. Use tools such as atop.
8. Review operating system logs for any errors, warnings, or high volumes of messages. Review logs such as /var/log/messages, /var/log/syslog, etc.
9. Review snapshots of process activity, and for the largest users of resources, review per thread activity. Use tools such as top -H.
10. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically. Review CPU steal time in tools such as vmstat, top, etc.
12. Linux on IBM Power CPUs:
   1. Test with the IBM Java parameter -Xnodfpbd
   2. Test with hardware prefetching disabled
   3. Test with idle power saver disabled
   4. Test with adaptive power boost enabled
   5. Test with dynamic power saver (favor performance) mode enabled
   6. Use 64-bit DMA adapter slots for network adapters
13. Linux on IBM System z CPUs:
   1. Use QUICKDSP for production guests

Also review the general topics in the Operating Systems chapter.

General

Check the system log for any warnings, errors, or repeated informational messages. The location or mechanism depends on the distribution. For example:

```
# less /var/log/messages
# less /var/log/syslog # Newer versions of Ubuntu
# journalctl # Newer versions of Fedora/RHEL
```

Query the help manual for a command:

```
$ man vmstat # By default, contents are sent to less
$ man -a malloc # There may be multiple manuals matching the name. Use -a to show all of them.
$ man -P cat vmstat # Use -P to send the output to something other than less. Note, if you pipe the output, it will figure that out and send things to stdout.
$ man -K vmstat # Search all manpages for a keyword
$ info libc # Some GNU programs offer more detailed documentation using the info command
```

Modifying Kernel Parameters

The kernel mounts a virtual filesystem in /proc/sys which exposes various kernel settings through pseudo files that can be read and (sometimes) written to get and set each value, respectively. For example, the following command gets the current value of the kernel's system wide limit of concurrently running threads/tasks:
$ sudo cat /proc/sys/kernel/threads-max
248744

Each of these pseudo files is documented in "man 5 proc": https://www.kernel.org/doc/man-pages/online/pages/man5/proc.5.html

If a value can be updated, simply echo the new value into the pseudo file:

$ echo 248745 > /proc/sys/kernel/threads-max
bash: /proc/sys/kernel/threads-max: Permission denied
$ sudo echo 248744 > /proc/sys/kernel/threads-max
bash: /proc/sys/kernel/threads-max: Permission denied

Notice that the user must have sufficient permissions, and simply prepending sudo is also not enough. The reason a simple "sudo echo" doesn't work is that this runs the echo command as root, but the output redirection occurs under the user's context. Therefore, you must use something like the tee command:

$ echo 248745 | sudo tee /proc/sys/kernel/threads-max
248745

This works but the change will be reverted on reboot. To make permanent changes, edit the /etc/sysctl.conf file as root. This lists key value pairs to be set on boot, separated by an equal sign. The key is the name of the pseudo file, with /proc/sys removed, and all slashes replaced with periods. For example, the same threads-max setting above would be added to /etc/sysctl.conf as:

kernel.threads-max=248745

Sysctl is also a command that can be run to print variables in a similar way to cat:

$ sudo sysctl kernel.threads-max
kernel.threads-max = 248745

Or to temporarily update variables similar to echo above and similar to the sysctl.conf line:

$ sudo sysctl -w kernel.threads-max=248746
kernel.threads-max = 248746

To list all current values from the system:

$ sudo sysctl -a | head
kernel.sched_child_runs_first = 0
kernel.sched_min_granularity_ns = 4000000
kernel.sched_latency_ns = 20000000

Finally, use the -p command to update kernel settings based on the current contents of /etc/sysctl.conf:

$ sudo sysctl -p
net.ipv4.ip_forward = 0
net.ipv4.conf.all_rp_filter = 1

The recommended way to edit kernel settings is to edit or add the relevant line in /etc/sysctl.conf and run $(sysctl -p). This will not only set the currently running settings, but it will also ensure that the new settings are picked up on reboot.
Processes

Query basic process information:

```bash
$ ps -elfyww | grep java
```

```
S  UID   PID  PPID  C PRI  NI  RSS  SZ  WCHAN  STIME  TTY  TIME  CMD
S  root  11386  1  17  80  0  357204  1244770  futex_  08:07  pts/2  00:00:30  java ... server1
```

Normally the process ID (PID) is the number in the fourth column, but the `-y` option (which adds the RSS column) changes PID to the third column. You can control which columns are printed and in which order using `-o`.

Note that even with the `-w` option or with a large `COLUMNS` envvar, the kernel limits the command line it stores to 4096 characters ([http://stackoverflow.com/a/199199/1293660](http://stackoverflow.com/a/199199/1293660)).

Central Processing Unit (CPU)

Query physical processor layout:

```bash
$ cat /proc/cpuinfo
```

```
processor : 0
model name : Intel(R) Core(TM) i7-3720QM CPU @ 2.60GHz
cpu cores : 4...
```

Query the current frequency of each CPU core (in Hz):

```bash
$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq
```

```
1200000
1200000
```

Query the NUMA layout:

```bash
$ numactl --hardware
```

```
available: 1 nodes (0)
node 0 cpus: 0 1 2 3 4 5 6 7
node 0 size: 16000 MB
node 0 free: 4306 MB
node distances:
  node 0:
    0: 10
```

CPU Speed

For maximum performance, ensure the scaling_governor is set to `performance` ([https://www.kernel.org/doc/Documentation/cpu-freq/user-guide.txt](https://www.kernel.org/doc/Documentation/cpu-freq/user-guide.txt)).

```bash
$ for i in /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor; do echo "performance" > $i; done
```

Consider disabling services such as $(cpuspeed), $(cpufreqd), $(powerd), etc.

Check the maximum frequency of each CPU core (in Hz):
```
$ for i in /sys/devices/system/cpu/cpu*/cpufreq/scaling_max_freq; do cat $i; done

vmstat

Query processor usage:
```
$ vmstat -t -n -SM 5 2
procs -----------memory---------- ---swap-- -----io---- --system--
-----cpu------ ---timestamp---
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
st
 0  0      0  10600    143   2271    0    0   114    24  150  623  3  1 93  3
 0    2014-02-10 08:18:37 PST
 0  0      0  10600    143   2271    0    0     2    24  679 1763  1  0 98  0
 0    2014-02-10 08:18:42 PST

To run vmstat in the background with a 5 second interval:
```
$ nohup vmstat -tn 5 > vmstat_${hostname}_${(date +"%Y%m%d_%H%M%S_%N").txt} &

Note: Some versions of Linux do not support the -t flag, so test this command first with $(vmstat -tn 5). If your version of vmstat does not support -t, just put a $(date) at the top of the file:
```
$ FILE=vmstat_${hostname}_${(date +"%Y%m%d_%H%M%S_%N").txt}; date > $FILE;
nohup vmstat -n 5 >> $FILE &

To stop collection, kill the vmstat process. For example:
```
$ pkill -f vmstat

vmstat notes:

- The first line is an average since reboot, so in most cases you should disregard it.
- The "r" column has had a confusing manual page in older releases. The newer description is more clear: "The "procs_running" line gives the total number of threads that are running or ready to run (i.e., the total number of runnable threads.)." ([https://www.kernel.org/doc/Documentation/filesystems/proc.txt](https://www.kernel.org/doc/Documentation/filesystems/proc.txt)).
- b: Average number of uninterruptible, blocked threads - usually I/O
- free, buff, cache: Equivalent to free command. “Total” free = free + buff + cache
- si/so: Swap in/out. bi/bo: Device blocks in/out
- id: Idle - Best place to look for CPU usage – substract 100 minus this column.
- Us=user CPU%, sy=system CPU%, wa=% waiting on I/O, st=% stolen by hypervisor

Ensure there are no errant processes using non-trivial amounts of CPU.

The kernel must be compiled with SMP enabled to utilize SMP CPUs. The sequence "SMP" will be in the $(uname -a) output if the kernel is SMP-aware.

Per Processor Utilization

Query per processor utilization:
```
$ mpstat -A 5 2
```
Some processors may have higher interrupt rates due to network card bindings.

top

top provides processor usage for the overall system and individual processes. Without arguments, it will periodically update the screen with updated information:

```
    top - 15:46:52 up 178 days, 4:53, 2 users, load average: 0.31, 0.08, 0.02
         Tasks:  77 total,   2 running,  74 sleeping,   1 stopped,   0 zombie
    Cpu(s): 24.6% us,  0.5% sy,  0.0% ni, 74.9% id,  0.0% wa,  0.0% hi,  0.0% si
    Mem:   5591016k total,  5416896k used,   174120k free,  1196656k buffers
    Swap:  2104472k total,    17196k used,  2087262k free,  2594884k cached
```

The CPU(s) row in this header section shows the CPU usage in terms of the following:

- **us**: Percentage of CPU time spent in user space.
- **sy**: Percentage of CPU time spent in kernel space.
- **ni**: Percentage of CPU time spent on low priority processes.
- **id**: Percentage of CPU time spent idle.
- **wa**: Percentage of CPU time spent in wait (on disk).
- **hi**: Percentage of CPU time spent handling hardware interrupts.
- **si**: Percentage of CPU time spent handling software interrupts.

```
    PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+
    COMMAND
          8502 user1     25   0  599m 466m 5212 R 99.9   8.5   0:23.92
```

The table represents the Process ID (PID), CPU usage percentage (%CPU), and process name (COMMAND) of processes using the most CPU. If the available CPU is 100% utilized, the availability to the Java process is being limited. In the case above, the Java process is using all the available CPU but is not contending with any other process. Therefore, the limiting performance factor is the CPU available to the machine.
If the total CPU usage is 100% and other processes are using large amounts of CPU, CPU contention is occurring between the processes, which is limiting the performance of the Java process.

Old Java Diagnostic Guide

Use the -b flag to run top in a batch mode instead of redrawing the screen every iteration. Use -d to control the delay between iterations and -n to control the number of iterations.

**Per-thread CPU Usage**

The output of top -H on Linux shows the breakdown of the CPU usage on the machine by individual threads. The top output has the following sections of interest:

```
  top - 16:15:45 up 21 days, 2:27, 3 users, load average: 17.94, 12.30, 5.52
  Tasks: 150 total, 26 running, 124 sleeping, 0 stopped, 0 zombie
  Cpu(s): 87.3% us, 1.2% sy, 0.0% ni, 27.6% id, 0.0% wa, 0.0% hi, 0.0% si
  Mem: 4039848k total, 3999776k used, 40072k free, 92824k buffers
  Swap: 2097144k total, 224k used, 2096920k free, 1131652k cached

  PID USER    PR  NI  VIRT  RES  SHR S %CPU   %MEM    TIME+ COMMAND
  31253 user1   16   0 2112m 2.1g 1764 R 37.0   53.2   0:39.89 java
  31249 user1   16   0 2112m 2.1g 1764 R 15.5   53.2   0:38.29 java
  31244 user1   16   0 2112m 2.1g 1764 R 13.6   53.2   0:40.05
```

**PID:** The thread ID. This can be converted into hexadecimal and used to correlate to the "native ID" in a javacore.txt file...

**S:** The state of the thread. This can be one of the following:

- R: Running
- S: Sleeping
- D: Uninterruptible sleep
- T: Traced
- Z: Zombie

**%CPU:** The percentage of a single CPU usage by the thread...

**TIME+:** The amount of CPU time used by the thread.

Note that the "Cpu(s)" line in the header of the output shows the percentage usage across all of the available CPUs, whereas the %CPU column represents the percentage usage of a
single CPU. For example, on a four-CPU machine the Cpu(s) row will total 100% and the
%CPU column will total 400%.

In the per-thread breakdown of the CPU usage shown above, the Java process is taking
approximately 75% of the CPU usage. This value is found by totaling the %CPU column
for all the Java threads (not all threads are shown above) and dividing by the number of
CPUs. The Java process is not limited by other processes. There is still approximately 25%
of the CPU idle. You can also see that the CPU usage of the Java process is spread
reasonably evenly over all of the threads in the Java process. This spread implies that no
one thread has a particular problem. Although the application is allowed to use most of the
available CPU, the fact that 25% is idle means that some points of contention or delay in
the Java process can be identified. A report indicating that active processes are using a
small percentage of CPU, even though the machine appears idle, means that the
performance of the application is probably limited by points of contention or process delay,
preventing the application from scaling to use all of the available CPU. If a deadlock is
present, the reported CPU usage for the Java process is low or zero. If threads are looping,
the Java CPU usage approaches 100%, but a small number of the threads account for all of
that CPU time. Where you have threads of interest, note the PID values because you can
convert them to a hexadecimal value and look up the threads in the javacore.txt file to
discover if the thread is part of a thread pool. In this way you gain an understanding of the
kind of work that the thread does from the thread stack trace in the javacore.txt file. For
example, the PID 31253 becomes 7A15 in hexadecimal. This value maps to the "native ID"
value in the javacore.txt file.

Old Java Diagnostic Guide

You can convert the thread ID into hexadecimal and search for it in a matching javacore.txt file on the
IBM JVM. For example, if the TID is 19511, convert 19511 to hexadecimal = 0x4C37. Search in
javacore for native ID:

"WebContainer : 1" (TID:0x0933CB00, sys_thread_t:0x09EC4774, state:CW, native
ID:0x00004C37) prio=5
java/text/FieldPosition$Delegate.formatted(FieldPosition.java:291(Compiled
Code))

Another technique to monitor per-thread CPU usage is to monitor the accumulated CPU time per
thread (TIME+) to understand which threads are using the CPUs.
The following command may be used to periodically gather the top 50 threads' CPU usage for the
entire machine:

$ cd /var/tmp/
$ nohup top -b -d 30 -H | grep -A 50 "top - " >> top_$(hostname)_$(date +%Y%m%d_%H%M%S_%N).out &

Note that this example of top -H may consume a significant amount of CPU because it must iterate
over all threads in the system.

To investigate a set of PIDs more directly, a command like the following may be useful, replace the
$PIDXs with your process IDs, and when looking at the top output, look at the second stanza:
while true; do for i in $PID1 $PID2 $PID3; do echo "Gathering data for PID $i at $(date)"; top -H -p $i -b -d 10 -n 2 > top_$(hostname)_$(date +%Y%m%d_%H%M%S_%N)_$i.out; kill -3 $i; done; echo "Sleeping at $(date)"; sleep 60; done

Load Average

"The first three fields in [/proc/loadavg] are load average figures giving the number of jobs in the run queue (state R) or waiting for disk I/O (state D) averaged over 1, 5, and 15 minutes." (http://man7.org/linux/man-pages/man5/proc.5.html)

A load average is usually reported as three numbers representing 1-minute, 5-minute, and 15-minute exponentially damped/weighted moving averages of the number of runnable and uninterruptible threads, usually recalculated every 5 seconds (https://en.wikipedia.org/wiki/Load_%28computing%29). If these numbers are greater than the number of CPU cores, then there may be cause for concern.

If capturing top -H during a time of a high load average does not show high CPU usage, then it is more likely caused by uninterruptible threads, which are usually waiting on I/O. There are reports that in some versions of Linux, there may be high numbers of threads waiting for I/O that may not all show up in CPU wait% or iostat. If CPU utilization does not correlate with load averages, review the number of threads in the "D" (uninterruptible) state.

See also: http://www.linuxjournal.com/article/9001?page=0,1

atop

Atop is an ASCII based live and historical system monitor (GNU General Public License): http://www.atoptool.nl/

Run without any options to do live monitoring:

```
$ atop
```

Includes crontab files to run atop in the background. Read a historical file:

```
# atop -r /var/log/atop/atop_20140908.1
```
Write atop data with a 10 second interval (Ctrl+C to stop):

```
atop -w atop.raw 10
```

Graph CPU usage of some process (replace the program name or PID in the first grep):

```
atop -PPRC -r atop.raw | grep java.*y$ | awk '{if(NR>1) {printf "%s %s,%d
", $4,$5,(($11+$12+$13)10)/$10} else print "Time,CPU%"} | 
  gnuplot -p -e "set timefmt '%Y/%m/%d %H:%M:%S'; set xtics out;set ytics out; set xdata time; set datafile sep ','; set grid; set style data lines; set format y '%.0f'; set format x '%H:%M:%S'; set key autotitle columnhead; plot '/dev/stdin' using 1:2; pause -1"
```

sar

sar is part of the sysstat package. It may be run periodically from a crontab in /etc/cron.d/sysstat and writes files to /var/log/sa/. You can report sar data textually on the system using the "sar" command:

```
$ sar -A | head
```
12:00:01 AM     CPU      %usr     %nice      %sys   %iowait    %steal
%irq     %soft    %guest     %idle
12:10:01 AM     all      0.86      0.00      0.59     0.15      0.00
0.00     0.00      0.00     98.41...

Some useful things to look at in sar:

- runq-sz
- plist-sz
- kmemused - kbuffers - kbcached

You can also visualize sar log files using ksar which is BSD license
(https://sourceforge.net/projects/ksar/):

![CPU for thumb1 on 09/28/06](image)

**nmon**

nmon was originally developed for AIX but has since been ported to Linux under the GPL license: http://nmon.sourceforge.net/pmwiki.php

One reason to use nmon on Linux is that the Java GUI nmon analyzer is a very powerful and flexible graphing application that accepts nmon data. For details, see the nmon section in the AIX chapter.

Start nmon for essentially unlimited collection with a 60 second interval:

```bash
# su
# cd /var/tmp/
```
Executing this command will start the nmon collector in the background, so explicitly putting it into the background (&) is not necessary. This will create a file with the name $HOST_\$STARTDAY_\$STARTTIME.nmon

Note that any errors starting nmon (such as file permissions writing to the specified directory) will go to nohup.out, so it is important to check nohup.out to make sure it started correctly. You can also run 'ps -elfx | grep nmon' to make sure it started.

When you want to stop nmon, run:

```
  # su
  # pkill -USR2 nmon
```

**collectl**

collectl is a comprehensive, open source, Linux monitoring tool created by RedHat: [http://collectl.sourceforge.net/](http://collectl.sourceforge.net/)

Collectl is a comprehensive performance data collection utility similar to sar. It is fine grained with low overhead and holistically collects all of the important kernel statistics as well as process data. Additionally, it is a very simple tool to collect very useful performance data.

While collectl is neither shipped nor supported by Red Hat at this time, it is a useful and popular utility frequently used by users and third party vendors.

[https://access.redhat.com/site/node/351143/](https://access.redhat.com/site/node/351143/)

**perf Profiler Tool**

perf is a profiler tool available since kernel version 2.6.

Query available CPU statistics:

```
  # perf list
```

```
List of pre-defined events (to be used in -e):
cpu-cycles OR cycles                                     [Hardware event]
instructions                                              [Hardware event]
cache-references                                          [Hardware event]
cache-misses                                              [Hardware event]...
```

Query CPU statistics for a process (use sleep X for some duration or without sleep X and Ctrl+C to stop):

```
  # perf stat -B -e cycles,cache-misses -p 11386 sleep 5
  Performance counter stats for process id '11386':
    20,810,324 cycles
    215,879 cache-misses
    5.000869037 seconds time elapsed
```
Sample CPU events for a process and then create a report:

```bash
# perf record -p 11386 sleep 5
# perf report
```

Query CPU statistics periodically:

```bash
# perf top
Samples: 5K of event 'cycles', Event count (approx.): 1581538113
  21.98% perf                                        [.] 0x000000000004bd30
  4.28% libc-2.12.so                                [.] __strcmp_sse42
```

Starting with IBM Java 7.1, the -Xjit:perfTool option may be specified to enhance the quality of symbols in perf output. Additional limitations may require the use of -Xlp:codecache:pagesize=4k.

**perf Flame Graphs**

Flame graphs are a great way to visualize CPU activity:

```
http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
```

```
# git clone https://github.com/brendangregg/FlameGraph
# cd FlameGraph
# perf record -F 99 -a -g -- sleep 60
# perf script | ./stackcollapse-perf.pl > out.perf-folded
# ./flamegraph.pl --width 600 out.perf-folded > perf-kernel.svg
```

**perf On-CPU Stack Sampling**

The $(perf record) command may be used to capture native stack traces on all CPUs at some frequency for some period of time. The following example captures all On-CPU stacks every 50ms for 10 seconds and writes the data to a file called perf.data:

```
sudo sh -c "date +%Y-%m-%d %H:%M:%S.%N %Z" >> perfdata_starttimes.txt;
cat /proc/uptime >> perfdata_starttimes.txt; perf record -T -F 99 -a -g -- sleep 60"
```

The frequency F may be converted to milliseconds (M) with the equation M=1000/F, so if you want to capture at a different millisecond frequency, use the equation F=1000/M. For example, to capture at 10ms frequency, F=1000/10, so the argument would be `-F 100`. It's generally a good idea to subtract 1 from F (e.g. `-F 99`) to avoid any coincidental sampling of application activity of the same frequency.

There is no way to change the output file name to something other than perf.data. If the file perf.data already exists, it is moved to `perf.data.old` before overwriting the existing file.
The reason for writing the date with millisecond precision into a separate file right before starting $(perf record) is that uptime may have drifted from wallclock time; therefore, it is not a reliable reflection of wallclock time (this is probably why the $(uptime) command only prints a relative amount) and stack tick offsets cannot be compared to the wallclock of uptime (e.g. $(date -d "1970-01-01 + $(date +%s) sec - $(cut -d ' ' -f1 </proc/uptime) sec" +"%F %T.%N UTC" > uptime.txt; date >> uptime.txt). When the $(perf) command reports the "captured on" wallclock time, it is simply looking at the creation time of the perf.data file (which usually occurs at the completion of the recording, so it's usually at the end of the sleep) which is a time_t, which is second precision, so the exact start time with millisecond precision is unavailable. This means that the only way to get millisecond precision wallclock time of a perf stack is to create a separate file that notes the wallclock time with millisecond accuracy right before starting perf.

Before recording, ensure that you have installed at least the kernel and glibc symbols (these are only used by the diagnostic tools to map symbols, so they do not change the function of the OS but they do use about 1GB of disk space).

For example, on RHEL:

1. Configure debuginfo repositories: [https://access.redhat.com/solutions/9907](https://access.redhat.com/solutions/9907)
2. sudo yum install kernel-debuginfo kernel-debuginfo-common glibc-debuginfo

If you are using IBM Java >= 7.1, then restart the JVM with the argument -Xjit:perfTool. The JIT will then write a file to /tmp/perf-${PID}.map which maps JIT-compiled method addresses to human-readable Java method names for the $(perf script) tool to use. For IBM Java < 7.1, use [https://github.com/kgibm/perf-map-agent/tree/ibmjava6](https://github.com/kgibm/perf-map-agent/tree/ibmjava6)

After the $(perf record) script has completed, process the data to human readable form:

```bash
chmod a+rw /tmp/perf-${PID}.map
sudo chown root:root /tmp/perf-${PID}.map
sudo perf script --header -I -f -F
  comm,cpu,pid,tid,time,event,ip,sym,dso,symoff > perfdata_$(date +%Y%m%d_%H%M%S_%N).txt
```

The perf script command might give various errors and warnings and they're usually about missing symbols and mapping files, which is generally expected (since it's sampling all processes on the box).

The time field is the number of seconds since boot (with microsecond precision after the decimal point), in the same format as the first column of /proc/uptime. The top of the perfdata file will include a timestamp when the $(perf record) command started writing the perf.data file (which usually occurs at the completion of the recording, so it's usually at the end of the sleep). For example:

```
# captured on: Tue Nov 13 11:48:03 2018
```

Therefore, one can approximate the wallclock time of each stack by taking the difference between the first stack's time field and the target stack's time field and adding that number of seconds to the captured time minus the sleep time. Unfortunately, this only gives second level resolution because the captured time only provides second level resolution. Instead, one can use the date printed into perfdata_starttimes.txt and add the difference in seconds to that date.

Example stack:

```
main 10840/10841 [006] 17020.130034: cycles:ppp:
  7f418d20727d Loop.main([Ljava/lang/String;)V_hot+0x189
```
The columns are:
1. Thread name
2. PID/TID
3. [CPUID]
4. Timestamp
5. perf event
6. Within each stack frame:
   1. Instruction pointer
   2. Method name+Offset
   3. Executable or shared object (or mapping file)

**System Tap (stap)**

System Tap simplifies creating and running kernel modules based on kprobes:


A simple "Hello World" script:

```bash
#!/usr/bin/stap
probe begin { println("Hello World") exit () }
```

Execute the script:

```
# stap helloworld.stp
```

For most interesting SystemTap scripts, the kernel development package and kernel symbols must be installed. Example scripts: [https://sourceware.org/systemtap/examples/](https://sourceware.org/systemtap/examples/)

Flame graphs are a great way to visualize CPU activity:
[http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html](http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html)

```
# stap -s 32 -D MAXBACKTRACE=1000000 -D MAXSTRINGLEN=4096 -D MAXMAPENTRIES=10240
-D MAXACTION=10000000 -D STP_OVERLOAD_THRESHOLD=5000000000 --all-modules
-ve 'global s; probe timer.profile { s[backtrace()] <<< 1; }'
probe end { foreach (i in s+) { print_stack(i);
printf("\t%d
", @count(s[i])); } } probe timer.s(60) { exit(); }'
> out.stap-stacks
`./stackcollapse-stap.pl out.stap-stacks > out.stap-folded`
# cat out.stap-folded | ./flamegraph.pl > stap-kernel.svg
```
Additional scripts:


**WAS Performance, Hang, or High CPU MustGather**

The WAS Performance, Hang, or High CPU MustGather is normally requested by IBM support: [http://www-01.ibm.com/support/docview.wss?uid=swg21115785](http://www-01.ibm.com/support/docview.wss?uid=swg21115785)

There are two scripts that may be used and by default, they run for a few minutes (thus they should be run during the issue):

1. perfMustGather.sh under "Collecting data using the WAIT Data Collector" which generates mustGather_RESULTS.tar.gz
2. linperf.sh under "Collecting data manually" which generates linperf_RESULTS.tar.gz

Both scripts collect similar data and they are run with the set of process IDs for the JVMs as parameters. Both scripts also request thread dumps through kill -3. In general, we recommend you use perfMustGather.sh because it is able to find the javacores that it requests and packages them in the resulting mustGather_RESULTS.tar.gz automatically. This must be manually done for linperf.sh.

**perfMustGather.sh**

Clone the problemdetermination git repository and run perfmustgather_vmstat.sh (requires Perl and gnuplot) in the same directory as the mustGather_RESULTS.tar.gz file:

```
$ git clone https://github.com/kgibm/problemdetermination
$ problemdetermination/scripts/linux/perfmustgather_vmstat.sh
```

This will generate a PNG file:
linperf.sh

Clone the problemdetermination git repository and run linperf_vmstat.sh (requires Perl and gnuplot) in the same directory as the linperf_RESULTS.tar.gz file:

```bash
$ git clone https://github.com/kgibm/problemetermination
$ problemdetermination/scripts/linux/linperf_vmstat.sh
```

This will generate a PNG file:
Intel Performance Counter Monitor (PCM)


$ make
$ sudo ./pcm.x

**EXEC**: instructions per nominal CPU cycle

**IPC**: instructions per CPU cycle

**FREQ**: relation to nominal CPU frequency='unhalted clock ticks'/'invariant timer ticks' (includes Intel Turbo Boost)

**AFREQ**: relation to nominal CPU frequency while in active state (not in power-saving C state)='unhalted clock ticks'/'invariant timer ticks while in C0-state' (includes Intel Turbo Boost)

**L3MISS**: L3 cache misses

**L2MISS**: L2 cache misses (including other core's L2 cache *hits*)
L3HIT : L3 cache hit ratio (0.00-1.00)
L2HIT : L2 cache hit ratio (0.00-1.00)
L3CLK : ratio of CPU cycles lost due to L3 cache misses (0.00-1.00), in some cases could be >1.0 due to a higher memory latency
L2CLK : ratio of CPU cycles lost due to missing L2 cache but still hitting L3 cache (0.00-1.00)
READ  : bytes read from memory controller (in GBytes)
WRITE  : bytes written to memory controller (in GBytes)
IO     : bytes read/written due to IO requests to memory controller (in GBytes); this may be an over estimate due to same-cache-line partial requests
TEMP   : Temperature reading in 1 degree Celsius relative to the TjMax temperature (thermal headroom): 0 corresponds to the max temperature

<table>
<thead>
<tr>
<th>Core (SKT)</th>
<th>EXEC</th>
<th>IPC</th>
<th>FREQ</th>
<th>AFREQ</th>
<th>L3MISS</th>
<th>L2MISS</th>
<th>L3HIT</th>
<th>L2HIT</th>
<th>L3CLK</th>
<th>L2CLK</th>
<th>READ</th>
<th>WRITE</th>
<th>IO</th>
<th>TEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.32</td>
<td>0.04</td>
<td>0.54</td>
<td>456 K</td>
<td>649 K</td>
<td>0.30</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.84</td>
<td>0</td>
<td>0.07</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0.01</td>
<td>0.54</td>
<td>0.02</td>
<td>0.46</td>
<td>286 K</td>
<td>412 K</td>
<td>0.31</td>
<td>0.31</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.91</td>
<td>0</td>
<td>0.08</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>65</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.00</td>
<td>0.45</td>
<td>0.01</td>
<td>0.47</td>
<td>106 K</td>
<td>119 K</td>
<td>0.11</td>
<td>0.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.29</td>
<td>0</td>
<td>0.03</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0.02</td>
<td>0.81</td>
<td>0.03</td>
<td>0.54</td>
<td>524 K</td>
<td>598 K</td>
<td>0.12</td>
<td>0.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.21</td>
<td>0</td>
<td>0.03</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0.01</td>
<td>0.67</td>
<td>0.02</td>
<td>0.46</td>
<td>229 K</td>
<td>264 K</td>
<td>0.13</td>
<td>0.20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.98</td>
<td>0</td>
<td>0.03</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.25</td>
<td>0.01</td>
<td>0.47</td>
<td>216 K</td>
<td>224 K</td>
<td>0.04</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.86</td>
<td>0</td>
<td>0.02</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0.00</td>
<td>0.15</td>
<td>0.00</td>
<td>0.46</td>
<td>18 K</td>
<td>19 K</td>
<td>0.02</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.42</td>
<td>0</td>
<td>0.01</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0.00</td>
<td>0.34</td>
<td>0.00</td>
<td>0.47</td>
<td>45 K</td>
<td>46 K</td>
<td>0.02</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.69</td>
<td>0</td>
<td>0.01</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-------------------------------
<table>
<thead>
<tr>
<th>SKT</th>
<th>EXEC</th>
<th>IPC</th>
<th>FREQ</th>
<th>AFREQ</th>
<th>L3MISS</th>
<th>L2MISS</th>
<th>L3HIT</th>
<th>L2HIT</th>
<th>L3CLK</th>
<th>L2CLK</th>
<th>READ</th>
<th>WRITE</th>
<th>IO</th>
<th>TEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.01</td>
<td>0.53</td>
<td>0.02</td>
<td>0.50</td>
<td>1884 K</td>
<td>2334 K</td>
<td>0.19</td>
<td>0.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.07</td>
<td>0</td>
<td>0.05</td>
<td>0.18</td>
<td>0.02</td>
<td>0.04</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-------------------------------
<table>
<thead>
<tr>
<th>TOTAL</th>
<th>EXEC</th>
<th>IPC</th>
<th>FREQ</th>
<th>AFREQ</th>
<th>L3MISS</th>
<th>L2MISS</th>
<th>L3HIT</th>
<th>L2HIT</th>
<th>L3CLK</th>
<th>L2CLK</th>
<th>READ</th>
<th>WRITE</th>
<th>IO</th>
<th>TEMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>0</td>
<td>0.01</td>
<td>0.53</td>
<td>0.02</td>
<td>0.50</td>
<td>1884 K</td>
<td>2334 K</td>
<td>0.19</td>
<td>0.21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.07</td>
<td>0</td>
<td>0.05</td>
<td>0.18</td>
<td>0.02</td>
<td>0.04</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Instructions retired: 167 M ; Active cycles: 317 M ; Time (TSC): 2597 Mticks ; C0 (active,non-halted) core residency: 3.03 %
C1 core residency: 4.92 %; C3 core residency: 1.98 %; C6 core residency: 0.09 %; C7 core residency: 89.97 %;
C2 package residency: 6.29 %; C3 package residency: 4.29 %; C6 package residency: 4.51 %; C7 package residency: 57.55 %;

PHYSICAL CORE IPC : 1.06 => corresponds to 26.41 %
utilization for cores in active state
Instructions per nominal CPU cycle: 0.02 => corresponds to 0.40 % core utilization over time interval
Physical Memory (RAM)


$ cat /proc/meminfo

MemTotal:       15943596 kB
MemFree:         4772348 kB
Buffers:          305280 kB
Cached:          8222008 kB
Slab:             369028 kB
AnonPages:     5397004 kB...

On newer versions of Linux, use the "Available" statistics to determine the approximate amount of RAM that's available for use for programs:

Many load balancing and workload placing programs check /proc/meminfo to estimate how much free memory is available. They generally do this by adding up "free" and "cached", which was fine ten years ago, but is pretty much guaranteed to be wrong today. It is wrong because Cached includes memory that is not freeable as page cache, for example shared memory segments, tmpfs, and ramfs, and it does not include reclaimable slab memory, which can take up a large fraction of system memory on mostly idle systems with lots of files. Currently, the amount of memory that is available for a new workload, without pushing the system into swap, can be estimated from MemFree, Active(file), Inactive(file), and SReclaimable, as well as the "low" watermarks from /proc/zoneinfo. However, this may change in the future, and user space really should not be expected to know kernel internals to come up with an estimate for the amount of free memory. It is more convenient to provide such an estimate in /proc/meminfo. If things change in the future, we only have to change it in one place.

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=34e431b0ae398fc54ea69ff85ec700722c9da773

Notes:

- Physical memory used ~= MemTotal - MemFree - Buffers - Cached
- AnonPages ~= The sum total of virtual memory allocations (e.g. malloc, mmap, etc.) by currently running processes. This is roughly equivalent to summing the RSS column in $(ps -eww -o pid,rss) (although RSS pages reported in $(ps) may be shared across processes):
  $ ps -eww -o pid,rss | tail -n+2 | awk '{print $2}' | paste -sd+ | bc

Determine the current level of memory utilization using the top utility.

$ top

top - 19:41:35 up 195 days, 8:48, 1 user, load average: 0.00, 0.00, 0.00
Tasks:  70 total,  1 running,  69 sleeping,  0 stopped,  0 zombie
Cpu(s):  0.1% us,  0.0% sy,  0.0% ni, 99.9% id,  0.0% wa,  0.0% hi,  0.0% si
Mem:      5591016k total, 4538132k used, 1052884k free, 1228700k buffers
Swap:  2104472k total,  17392k used, 2087080k free, 2293776k
Mem  Physical memory (RAM) in total, used and free amounts
Swap  Swap or paging space in total, used and free amounts
cached  Amount of physical memory used for disk caching
buffers  Amount of physical memory used for input/output queues

The Linux operating system tries to use as much of the physical memory (RAM) as possible for the running applications as well as for the caching of file systems and input/output queues. The physical memory used for file caching (cached) and input/output queues (buffers) is released to the applications if required. The amount of physical memory available for additional application use is the free Mem value, added to the values for buffers and cached. In the case above, this value is:

1052884k (free) + 1228700k (buffers) + 2293776k (cached) = 4575360k = 4468MB = 4.4GB

If there is very little or no free Mem and the values for buffers and cached are also very low, you need more physical memory. The amount of physical memory must be increased by at least the amount of Swap used, and preferably by greater than this to allow for file system caching and input/output queue buffers.

Old Java Diagnostic Guide

cgroups can be used to limit memory usage of a process or set of processes:
https://www.kernel.org/doc/Documentation/cgroups/memory.txt

Per-process Memory Usage

Use the ps command to show the resident and virtual sizes of a process:

$ ps -eww -o pid,rss,vsz,command
   PID   RSS    VSZ COMMAND
32665 232404 4777744 java ... server1

Resident memory pages may be shared across processes. The file /proc/$PID/smaps includes a "Pss" line for each virtual memory area which is the proportional set size, which is a subset of RSS, and tries to take into account shared resident pages.

File cache

By default, Linux aggressively caches content such as parts of files in memory. Most or all of this physical memory usage will be pushed out of memory if program demands require it; therefore, in general, to understand physical memory usage, subtract "cached" (and "buffers") from used memory:

There is a way to flush the file cache from physical memory. Although this is generally not required, it may be useful before running an iteration of a stress test to ensure maximum comparability with previous runs:

$ sudo sync
$ echo 1 | sudo tee /proc/sys/vm/drop_caches

To free pagecache:

    echo 1 > /proc/sys/vm/drop_caches
To free reclaimable slab objects (includes dentries and inodes):
    echo 2 > /proc/sys/vm/drop_caches
To free slab objects and pagecache:
    echo 3 > /proc/sys/vm/drop_caches

This is a non-destructive operation and will not free any dirty objects. To increase the
number of objects freed by this operation, the user may run $(sync) prior to writing to
/proc/sys/vm/drop_caches. This will minimize the number of dirty objects on the system
and create more candidates to be dropped.

This file is not a means to control the growth of the various kernel caches (inodes, dentries,
pagecache, etc...) These objects are automatically reclaimed by the kernel when memory is
needed elsewhere on the system.

Use of this file can cause performance problems. Since it discards cached objects, it may
cost a significant amount of I/O and CPU to recreate the dropped objects, especially if they
were under heavy use. Because of this, use outside of a testing or debugging environment
is not recommended. (https://www.kernel.org/doc/Documentation/sysctl/vm.txt)

free

Query physical memory usage:

    $ free -m

```
    total   used   free   shared  buffers  cached
    Mem:    15569  10888  4681     0     298   8029
    -/+ buffers/cache:    2561  13008
    Swap:       0       0       0
```

In general, you want to look at the "-/+ buffers/cache" line because buffers and cache are not program memory.

/proc/meminfo

"meminfo: Provides information about distribution and utilization of memory."
(https://www.kernel.org/doc/Documentation/filesystems/proc.txt)

Example (only showing first few lines):

    $ cat /proc/meminfo | head -4
    MemTotal:  15943596 kB
    MemFree:    2870172 kB
    Buffers:    346644 kB
    Cached:     9689544 kB
To find how much memory is used by programs and kernel:

\[ \text{MemUsed} = \text{MemTotal} - \text{MemFree} - \text{Buffers} - \text{Cached} \]

In the above example = 2.89GB

**Paging**

When the physical memory is full, paging (also known as swapping) occurs to provide additional memory. Paging consists of writing the contents of physical memory to disk, making the physical memory available for use by applications. The least recently used information is moved first. Paging is expensive in terms of performance because, when required information is stored on disk it must be loaded back into physical memory, which is a slow process.

Where paging occurs, Java applications are impacted because of garbage collection. Garbage collection requires every part of the Java heap to be read. If any of the Java heap has been paged out, it must be paged back when garbage collection runs, slowing down the garbage collection process.

The `vmstat` output shows whether paging was taking place when the problem occurred. `vmstat` output has the following format:

```
procs -----------memory---------- ---swap--  -----io---- --system--
----cpu----
  r  b   swpd   free    buff   cache   si   so    bi    bo   in    cs us sy
  id   wa
 0  0  17196  679860  1196656  2594884  0  0    1    4     0  0  0  0
100  0
 0  0  17196  679868  1196656  2594884  0  0    0   40   1012  43  0  0
100  0
 0  0  17196  679992  1196656  2594884  0  0    0     3  1004  43  0  0
100  0
```

The columns of interest are... `si` and `so` (swap in and swap out) columns for Linux. Nonzero values indicate that paging is taking place.

**Shared Memory**

It may be necessary to tune the kernel's shared memory configuration for products such as databases ([https://www.kernel.org/doc/Documentation/sysctl/kernel.txt](https://www.kernel.org/doc/Documentation/sysctl/kernel.txt)).

- `/proc/sys/kernel/shmall`: The maximum amount of shared memory for the kernel to allocate.
- `/proc/sys/kernel/shmmmax`: The maximum size of any one shared memory segment.
- `/proc/sys/kernel/shmmni`: The maximum number of shared memory segments.

For example, set `kernel.shmmax=1073741824` in `/etc/sysctl.conf` and apply with `sysctl -p`.

**GLIBC malloc**

In recent kernels, the text is at the bottom, stack at the top, and mmap/heap sections grow towards each other in a shared space (although they cannot overlap). By default, the malloc implementation in glibc (which was based on ptmalloc, which in turn was based on dlmalloc) will allocate into either the native
heap (sbrk) or mmap space, based on various heuristics and thresholds: If there's enough free space in
the native heap, allocate there. Otherwise, if the allocation size is greater than some threshold (slides
between 128KB and 32/64MB based on various factors [1]), allocate a private, anonymous mmap
instead of native heap (mmap isn't limited by ulimit -d) (http://man7.org/linux/man-
pages/man3/mallopt.3.html)

In the raw call of sbrk versus mmap, mmap is slower because it must zero the range of bytes

**MALLOC_ARENA_MAX**

Starting with glibc 2.11 (for example, customers upgrading from RHEL 5 to RHEL 6), by default, when
glibc malloc detects mutex contention (i.e. concurrent mallocs), then the native malloc heap is broken
up into sub-pools called arenas. This is achieved by assigning threads their own memory pools and by
avoiding locking in some situations. The amount of additional memory used for the memory pools (if
any) can be controlled using the environment variables MALLOC_ARENA_TEST and
MALLOC_ARENA_MAX. MALLOC_ARENA_TEST specifies that a test for the number of cores is
performed once the number of memory pools reaches this value. MALLOC_ARENA_MAX sets the
maximum number of memory pools used, regardless of the number of cores.

The default maximum arena size is 1MB on 32-bit and 64MB on 64-bit. The default maximum number
of arenas is the number of cores multiplied by 2 for 32-bit and 8 for 64-bit.

This can increase fragmentation because the free trees are separate.

In principle, the net performance impact should be positive of per thread arenas, but testing different
arena numbers and sizes may result in performance improvements depending on your workload.

You can revert the arena behavior with the environment variable MALLOC_ARENA_MAX=1

**OOM Killer**

If /proc/sys/vm/overcommit_memory is set to 0 (the default), then the Linux kernel will allow memory
overcommit. If RAM and swap space become exhausted, the Linux oom-killer will send a SIGKILL (9)
signal to processes until sufficient space is freed:

```plaintext
By default, Linux follows an optimistic memory allocation strategy. This means that when
malloc() returns non-NULL there is no guarantee that the memory really is available. In
case it turns out that the system is out of memory, one or more processes will be killed by
the OOM killer (https://www.kernel.org/doc/man-pages/online/pages/man3/malloc.3.html,
https://www.kernel.org/doc/Documentation/sysct1/vm.txt,
```

The SIGKILL signal cannot be caught, blocked, or ignored by processes, and no process core dump is

When the OOM killer is invoked, a message is written to the system log. Recent versions include a list
of all tasks and their memory usage. For example:

```plaintext
kernel: Out of Memory: Killed process 123 (someprocess).
kernl: someprocess invoked oom-killer: gfp_mask=0xabc, order=0,
oom_score_adj=0
```
If /proc/sys/vm/panic_on_oom is set to 1, then a kernel panic will be produced when the OOM killer is triggered and the system is rebooted. Creating a dump on a panic requires configuring kdump:

The kernel decides which process to kill based on various heuristics and per-process configuration (see chapter 3 section 1 in https://www.kernel.org/doc/Documentation/filesystems/proc.txt). A process may be excluded from the oom-killer by setting its oom_score_adj to -1000:

```bash
$ echo -1000 > /proc/${PID}/oom_score_adj
```

The OOM killer may be disabled. For example, set vm.overcommit_memory=2 and vm.overcommit_ratio=100 in /etc/sysctl.conf and apply with sysctl -p. In this case, malloc will return NULL when there is no memory and available. Many workloads can't support such configurations because of high virtual memory allocations.

While there is considerable philosophical debate about swap, consider disabling swap, setting vm.panic_on_oom=1, and configuring kernel vmcore dumps with process-level virtual address space information to avoid swap thrashing situations and reduce downtime, whilst analyzing post-mortem vmcores for excessive memory usage, leaks, or undersizing.

To disable swap, use $(swapoff -a) to immediately disable swap partitions, and then remove any swap partitions from /etc/fstab for future reboots.

Example of configuring kdump on RHEL:

1. Configure, start, and enable the crash kernel/kdump: http
2. Size the amount of RAM for the crash kernel correctly: http
   1. Change /etc/kdump.conf to ensure makedumpfile uses `-d 23,31` so that process virtual address space information is dumped for each user process (command line arguments, virtual memory, etc.).
3. Set vm.panic_on_oom=1 in /etc/sysctl.conf
4. Install the kernel-debuginfo, kernel-debuginfo-common, glibc-debuginfo, and glibc-debuginfo-common packages: https://access.redhat.com/solutions/9907
5. Install the RedHat crash utility: http

For more information on how to produce and analyze kernel vmcores, see https://publib.boulder.ibm.com/httpserv/cookbook/InterConnect2016_7393.pdf

/proc/sys/vm/swappiness

"This control is used to define how aggressive the kernel will swap memory pages. Higher values will increase aggressiveness, lower values decrease the amount of swap. The default value is 60."

swappiness, is a parameter which sets the kernel's balance between reclaiming pages from the page cache and swapping out process memory. The reclaim code works (in a very simplified way) by calculating a few numbers:
• The "distress" value is a measure of how much trouble the kernel is having freeing memory. The first time the kernel decides it needs to start reclaiming pages, distress will be zero; if more attempts are required, that value goes up, approaching a high value of 100.
• mapped_ratio is an approximate percentage of how much of the system's total memory is mapped (i.e. is part of a process's address space) within a given memory zone.
• vm_swappiness is the swappiness parameter, which is set to 60 by default.

With those numbers in hand, the kernel calculates its "swap tendency":

\[ \text{swap\_tendency} = \frac{\text{mapped\_ratio}}{2} + \text{distress} + \text{vm\_swappiness} \]

If swap_tendency is below 100, the kernel will only reclaim page cache pages. Once it goes above that value, however, pages which are part of some process's address space will also be considered for reclaim. So, if life is easy, swappiness is set to 60, and distress is zero, the system will not swap process memory until it reaches 80% of the total. Users who would like to never see application memory swapped out can set swappiness to zero; that setting will cause the kernel to ignore process memory until the distress value gets quite high.

http://lwn.net/Articles/83588/

A value of 0 tells the kernel to avoid paging program pages to disk as much as possible. A value of 100 encourages the kernel to page program pages to disk even if filecache pages could be removed to make space.

Note that this value is not a percentage of physical memory, but as the above example notes, it is a variable in a function. If distress is low and the default swappiness of 60 is set, then program pages may start to be paged out when physical memory exceeds 80% usage (where usage is defined as usage by program pages). Which is to say, by default, if your programs use more than 80% of physical memory, the least used pages in excess of that will be paged out.

This may be adversely affecting you if you see page outs but filecache is non-zero. For example, in vmstat, if the "so" column is non-zero (you are paging out) and the "cache" column is a large proportion of physical memory, then the kernel is avoiding pushing those filecache pages out as much as it can and instead paging program pages. In this case, either reduce the swappiness or increase the physical memory. This assumes the physical memory demands are expected and there is no leak.

In general, for Java-based workloads which have light disk file I/O, set vm.swappiness=0 in /etc/sysctl.conf and apply with sysctl -p.

Note that recent versions of the Linux kernel (generally >= 3.5) have made vm.swappiness=0 more aggressive in avoiding swapping out anonymous pages. Some prefer to use vm.swappiness=1 to retain the old behavior of a slight preference for some swapping of anonymous pages under memory pressure. For the purposes of the above recommendations for Java-based workloads which have light disk file I/O, it's preferable to set vm.swappiness=0.

http://lwn.net/Articles/83588/
pdflush

The pdflush process writes dirty file page cache entries to disk asynchronously.

/proc/sys/vm/dirty_writeback_centisecs controls the frequency pdflush awakes and
/proc/sys/vm/dirty_expire_centiseconds controls the threshold at which a dirty page is judged that it
needs to be written by a run of pdflush (or if memory is low, judged with
/proc/sys/vm/dirty_background_ratio). If the total size of dirty pages as a proportion of physical
memory exceeds /proc/sys/vm/dirty_ratio, processes write to disk synchronously.

If system I/O activity is heavy but bursty and this causes problems, consider reducing the above
variables, first starting with dirty_background_ratio (e.g. 3), followed by dirty_ratio (e.g. 15), followed
by dirty_expire_centiseconds (e.g. 500), followed by dirty_writeback_centisecs (e.g. 100).

For example, set vm.dirty_background_ratio=3 in /etc/sysctl.conf and apply with sysctl -p

Input/Output (I/O)

Unless tracking file and directory access times is required, use the noatime and nodiratime flags (or
consider relatime) when mounting filesystems to remove unnecessary disk activity

Query disk usage:

```
$ df -h
Filesystem                       Size  Used Avail Use% Mounted on
/dev/mapper/vg_lifeboat-lv_root  385G  352G   14G  97% /
/tmpfs                            7.7G  628K  7.7G   1% /dev/shm
/dev/sda1                        485M   97M  363M  22% /boot
```

Query filesystem information:

```
$ stat -f /
File: "/"
   ID: 2975a4f407cfa7e5 Namelen: 255     Type: ext2/ext3
   Block size: 4096       Fundamental block size: 4096
   Blocks: Total: 100793308  Free: 8616265    Available: 3496265
   Inodes: Total: 25600000   Free: 20948943
```

Query disk utilization:

```
$ iostat -xm 5 2
Linux 2.6.32-358.11.1.el6.x86_64 (oc2613817758.ibm.com) 02/07/2014
   _x86_64_     (8 CPU)
      avg-cpu:  %user   %nice %system %iowait %steal   %idle
               1.17    0.00    0.55    0.25    0.00   98.03
Device:     rrqm/s  wrqm/s  r/s  w/s  rMB/s  wMB/s  avgrq-sz
   sda         0.17  17.13   1.49  3.63  0.05   0.08   50.69
   dm-0        0.00   0.00  1.48  20.74  0.05   0.08   11.59
   dm-1        7.46  335.73  0.92  2.05  0.05   0.08   11.68
   dm-2        0.00   0.00  1.48  20.57  0.05   0.08   11.68
   dm-3        0.00   0.00  1.48  20.57  0.05   0.08   11.68
```
Networking

Query interfaces:

$ netstat -i

Kernel Interface table

Iface     MTU  Met  RX-OK  RX-ERR  RX-DRP  RX-OVR  TX-OK  TX-ERR  TX-DRP  TX-OVR  Flg
eth0      1500  0     0      0      0      0      0      0      0      0  0 BMU
lo        16436 0  3162172 0      0      0  3162172 0      0      0  0 LRU
tun0      1362  0    149171 0      0      0   150329 0      0      0  0 MOPRU
virbr0    1500  0    43033  0      0      0    63937 0      0      0  0 BMRU
virbr1    1500  0      0      0      0      0     124 0      0      0  0 BMRU
wlan0     1500  0  1552613 0      0      0   704346 0      0      0  0 BMRU

Use netstat to collect a snapshot of network activity: netstat -antop

Example:

$ sudo netstat -antop

Active Internet connections (servers and established)
Proto  Recv-Q Send-Q  Local Address                      Foreign Address  State PID/Program name    Timer
tcp    0      0  0.0.0.0:6000                0.0.0.0:* LISTEN  3646/Xorg           off (0.00/0/0)
tcp    0      0 10.20.117.232:46238        10.20.54.72:80 ESTABLISHED 4140/firefox      off (0.00/0/0)
tcp    0      0 10.20.133.78:35370          10.20.33.79:1352 ESTABLISHED 5441/notes          keepalive (3542.42/0/0)
tcp    0      1 ::ffff:10.20.133.78:49558   ::ffff:10.20.52.206:52311 SYN_SENT 3502/BESClient  on (7.65/4/0)

The -o parameter adds the Timer column which will show various timers. For example, the first number
before the slash for timewait indicates how many seconds until the socket will be cleared.

Query network interface statistics:

$ netstat -s

Ip:
5033261 total packets received
89926 forwarded
0 incoming packets discarded
4223478 incoming packets delivered
4202714 requests sent out
38 outgoing packets dropped
2 dropped because of missing route
26 reassemblies required
13 packets reassembled ok

Tcp:
15008 active connections openings
248 passive connection openings
611 failed connection attempts
160 connection resets received
4 connections established
4211392 segments received
4093580 segments send out
8286 segments retransmited
0 bad segments received.
3055 resets sent...

Ping a remote host. In general, and particularly for LANs, ping times should be less than a few hundred milliseconds with little standard deviation.

$ ping -n 10.20.30.1
PING 10.20.30.1 (10.20.30.1) 56(84) bytes of data.
64 bytes from 10.20.30.1: icmp_seq=1 ttl=250 time=112 ms
64 bytes from 10.20.30.1: icmp_seq=2 ttl=250 time=136 ms
64 bytes from 10.20.30.1: icmp_seq=3 ttl=250 time=93.8 ms
64 bytes from 10.20.30.1: icmp_seq=4 ttl=250 time=91.6 ms

Since kernel 2.6.18, the current and maximum sizes of the socket backlog on a connection are reported in the Recv-Q and Send-Q columns, respectively, for listening sockets:

**Recv-Q**
Established: The count of bytes not copied by the user program connected to this socket.

Listening: Since Kernel 2.6.18 this column contains the current syn backlog.

**Send-Q**
Established: The count of bytes not acknowledged by the remote host.

Listening: Since Kernel 2.6.18 this column contains the maximum size of the syn backlog.

**ls/of**
Running lsof:

```
ls/of
```

Running lsof if only interested in network (some of the flags imply not showing regular files):

```
ls/of -Pnl
```

Last command but grouping by TCP socket connection states:

```
ls/of -Pnl | grep "TCP " | awk '{print $(NF)}' | sort | uniq -c
```
Networked Filesystems (NFS)

NFS may be monitored with tools such as nfsiostat (https://www.kernel.org/doc/man-pages/online/pages/man8/nfsiostat.8.html). For example:

```bash
$ FILE=nfsiostat_${hostname}_${date::-%Y%m%d_%H%M%S_%N}.txt; date > $FILE; 
nohup stdbuf --output=L nfsiostat 300 >> $FILE &
```

Note: Without using `stdbuf`, older versions of nfsiostat do not flush output when stdout is redirected, so output to the file may be delayed.

For example:

```
nfs.example.com:/path mounted on /path:
op/s      rpc bklog
  189.86    0.00
read:      avg RTT (ms)       avg exe (ms)       kB/s       kB/op       retrans
    3.755       60.772       16.186       4 (0.0%)
write:     avg RTT (ms)       avg exe (ms)       kB/s       kB/op       retrans
    148.911      446.987       3.002       22 (0.0%)
```

ethtool

ethtool may be used to query network driver and hardware settings. For example, to query the ring buffers:

```
# ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX:  2040
RX Mini:  0
RX Jumbo:  8160
TX:  255
Current hardware settings:
RX:  255
RX Mini:  0
RX Jumbo:  0
TX:  255
```

Socket Buffers

The default receive buffer size for all network protocols is net.core.rmem_default (https://www.kernel.org/doc/man-pages/online/pages/man7/socket.7.html). The default receive buffer size for TCP sockets (for both IPv4 and IPv6) is the second value of net.ipv4.tcp_rmem (https://www.kernel.org/doc/man-pages/online/pages/man7/tcp.7.html, https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt). These values may be overridden by an explicit call to setsockopt(SO_RCVBUF) which will set the receive buffer size to two times the requested value (https://www.kernel.org/doc/man-pages/online/pages/man7/socket.7.html). The default or requested receive buffer size is limited by net.core.rmem_max and, in the case of TCP, the third
value (max) in net.ipv4.tcp_rmem.

Starting with Linux 2.4.17 and 2.6.7, the kernel auto-tunes the TCP receive buffer by default. This is controlled with the property tcp_moderate_rcvbuf (https://www.kernel.org/doc/man-pages/online/pages/man7/tcp.7.html). If auto-tuning is enabled, the kernel will start the buffer at the default and modulate the size between the first (min) and third (max) values of net.ipv4.tcp_rmem, depending on memory availability. In general, the min should be set quite low to handle the case of physical memory pressure and a large number of sockets.

The default send buffer size for all network protocols is net.core.wmem_default (https://www.kernel.org/doc/man-pages/online/pages/man7/socket.7.html). The default send buffer size for TCP sockets (for both IPv4 and IPv6) is the second value of net.ipv4.tcp_wmem (https://www.kernel.org/doc/man-pages/online/pages/man7/tcp.7.html, https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt). These values may be overridden by an explicit call to setsockopt(SO_SNDBUF) which will set the send buffer size to two times the requested value (https://www.kernel.org/doc/man-pages/online/pages/man7/socket.7.html). The default or requested send buffer size is limited by net.core.wmem_max and, in the case of TCP, the third value (max) in net.ipv4.tcp_wmem.

In general, the maximum socket receive and send buffer sizes should be greater than the average bandwidth delay product (see the Operating Systems chapter).

For example, consider setting values similar to the following in /etc/sysctl.conf and running $(sysctl -p):

```plaintext
net.core.rmem_default=1048576
net.core.wmem_default=1048576
net.core.rmem_max=16777216
net.core.wmem_max=16777216
net.ipv4.tcp_rmem=4096 1048576 16777216
net.ipv4.tcp_wmem=4096 1048576 16777216
```

Both receive and send TCP buffers (for both IPv4 and IPv6) are regulated by net.ipv4.tcp_mem (https://www.kernel.org/doc/man-pages/online/pages/man7/tcp.7.html). tcp_mem is a set of three numbers - low, pressure, and high - measured in units of the system page size ($(getconf PAGESIZE)). When the number of pages allocated by receive and send buffers is below `low`, TCP does not try to reduce its buffers' memory usage. When the number of pages exceeds `pressure`, TCP tries to reduce its buffers' memory usage. The total buffers' memory usage page may not exceed the number of pages specified by `high`. In general, these values are set as some proportions of physical memory, taking into account program/computational demands. By default, Linux sets these to proportions of RAM on boot. Query the value with sysctl and multiply the middle number by the page size (often 4096) and this is the number of bytes at which point the OS may start to trim TCP buffers.

Tuning done for SPECj: http://www.spec.org/jEnterprise2010/results/res2013q2/jEnterprise2010-20130402-00042.html#JEE_AppServer_HW_0

**Emulating Network Behaviors**

netem is a network emulation component of the traffic control (tc) suite. For example, to emulate a 100ms delay on all packets:

```plaintext
# tc qdisc add dev ${INTERFACE} root netem delay 100ms
```
TCP Congestion Control

The TCP specifications require that a TCP sender implements a congestion window (cwnd) to regulate how many packets are outstanding at any point in time. The congestion window is in addition to any constraints advertised by the receiver window size. The default congestion algorithm is cubic. A space-delimited list of available congestion algorithms may be printed with:

```
$ sysctl net.ipv4.tcp_available_congestion_control
net.ipv4.tcp_available_congestion_control = cubic reno
```

Additional congestion control algorithms, often shipped but not enabled, may be enabled with modprobe. For example:

```
# modprobe tcp_hybla
```

The current congestion control algorithm may be dynamically updated with:

```
# sysctl -w net.ipv4.tcp_congestion_control=hybla
```

An example symptom of a congestion control algorithm limiting throughput is when a sender has queued X bytes to the network, the current receive window is greater than X, but less than X bytes are sent before waiting for ACKs from the receiver. In one case, changing to hybla, which is designed for high latency connections, improved performance. In another case, on a low latency network, changing to hybla decreased performance. Another commonly used algorithm is htcp.

The congestion window is not advertised on the network but instead lives within memory on the sender. To query the congestion window, use the $(ss) command and search for the `cwnd` value:

```
$ ss -i
State      Recv-Q Send-Q      Local Address:Port          Peer Address:Port
ESTAB      0      0            10.20.30.254:47768        10.20.30.40:http
cubic wscale:0,9 rto:266 rtt:66.25/25.25 ato:40 cwnd:10 send 1.7Mbps
rcv_space:14600
```

The default congestion window size (initcwnd) may be changed by querying the default route and using the change command with initcwnd added. For example:

```
# ip route show | grep default
default via 10.20.30.1 dev wlan0  proto static
# ip route change default via 10.20.30.1 dev wlan0  proto static initcwnd 10
```

The default receive window size (initrwnd) may be changed in a similar way.

Starting with kernel version 2.6.18, by default, a socket's congestion window will be reduced when idle. If using persistent connections or connections with periods of non-trivial delays, consider setting the following option in /etc/sysctl.conf and running $(sysctl -p):

```
net.ipv4.tcp_slow_start_after_idle=0
```

Changing the MTU

If all components in a network path support larger MTU (sometimes called "jumbo frames") and if this setting is enabled on these devices, then an MTU line may be added to /etc/sysconfig/network-
scripts/ifcfg-${INTERfACE} and the network service restarted to utilize the larger MTU. For example:
MTU=9000

TCP Reordering
In some benchmarks, changing the values of net.ipv4.tcp_reordering and net.ipv4.tcp_reordering
Other Network Configuration
To update the listen backlog, set net.core.somaxconn=511 in /etc/sysctl.conf and apply with sysctl -p
(https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tu
nelinux.html)
To increase the maximum incoming packet backlog, set net.core.netdev_max_backlog=300000 in
/etc/sysctl.conf and apply with sysctl -p
(https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tu
Each network adapter has an outbound transmission queue which limits the outbound TCP sending
rate. Consider increasing this by running "ifconfig ${DEVICE} txqueuelen 4096" on each device.
(http://www.ibm.com/support/knowledgecenter/SSFK3V_1.3.0/com.ibm.cluster.pe.v1r3.pe200.doc/am
101_tysfbpjp.htm)
Update the TCP Keepalive interval by setting net.ipv4.tcp_keepalive_intvl=15 in /etc/sysctl.conf and
apply with sysctl -p
(https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tu
nelinux.html)
Update the TCP Keepalive probe count by setting net.ipv4.tcp_keepalive_probes=5 in /etc/sysctl.conf
and apply with sysctl -p
(https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tu
nelinux.html)
tcpdump
Capture network packets using tcpdump (http://www-01.ibm.com/support/docview.wss?
uid=swg21175744).
Normally, tcpdump is run as root. For example, capture all traffic in files of size 100MB and up to 10
historical files (-C usually requires -Z):
$ (sudo nohup tcpdump -nn -v -i any -B 4096 -s 0 -C 100 -W 10 -Z root -w
capture_$(hostname)_$(date +"%Y%m%d_%H%M%S_%N").pcap &); sleep 2; sudo cat
nohup.out

If the traffic in question occurs on a single interface, it's better to replace `-i any` with the actual
interface name as this has less of a chance to confuse Wireshark than the `any` pseudo-interface.
If `-W 1` is specified, there will be just one file and it will overwrite at the beginning when rotating, so
it's usually better to use `-W 2` with half the desired `-C` to ensure having some history (e.g. if the
Page 78


problem is reproduced right after a rotation). If -W is not specified, the behavior is unclear with some testing showing strange behavior, so it's best to specify -W.

To stop the capture:

$ sudo pkill -INT tcpdump

Use Wireshark to analyze (covered in the Major Tools chapter).

Depending on the version of tcpdump, it may have a default snarflen (-s) which is very small or 65535. In the case of 65535, remember that most packets are much smaller than this (usually limited by the MTU), so 65535 is effectively unlimited.

Filter what traffic is captured using an expression (see the man page). For example, to only capture traffic coming into or going out of port 80:

$ sudo tcpdump -i any -B 4 -w capture.pcap "port 80"

In addition to using Wireshark, you may also dump the tcpdump on any Linux machine using the same tcpdump command. For example:

$ sudo tcpdump -A -n -nn -l -tttt -r capture.pcap

If you would like to only capture the TCP headers, then the best way to do this is to do a capture of representative traffic, then load in Wireshark, filter to tcp packets, sort by frame length and then take the smallest value and use this value N for -s. For example:

$ sudo tcpdump -s N ...

If you see X "packets dropped by kernel" > 0, continue increasing -B N (where N is in KB):

Packets that arrive for a capture are stored in a buffer, so that they do not have to be read by the application as soon as they arrive. On some platforms, the buffer's size can be set; a size that's too small could mean that, if too many packets are being captured and the snapshot length doesn't limit the amount of data that's buffered, packets could be dropped if the buffer fills up before the application can read packets from it, while a size that's too large could use more non-pageable operating system memory than is necessary to prevent packets from being dropped.


Kernel

Thread Stacks

Output /proc/pid/stack and /proc/pid/task/*/stack to review all kernel stacks.

Process Tracing

strace may be used to trace system calls that a process makes, and ltrace may be used to trace library calls that a process makes. This can be helpful in certain situations when there are low level delays such
as writing to disk (strace), or investigating library calls such as libc malloc calls (ltrace).

```sh
$ strace -f -tt -o outputfile.txt -p $PID
```

```sh
31113 11:43:15.724911 open("/home/user/somefile", O_WRONLY|O_CREAT|O_TRUNC|O_LARGEFILE, 0666) = 139
31113 11:43:15.725109 fstat64(139, {st_mode=S_IFREG|0664, st_size=0, ...}) = 0
31113 11:43:15.728881 write(139, "<!DOCTYPE html PUBLIC ",8192
<unfinished ...>
31113 11:43:15.729004 <... write resumed> ) = 8192
31113 11:43:15.729385 close(139 <unfinished ...>
31113 11:43:15.731440 <... close resumed> ) = 0
```

Thread ID, timestamp and call = RESULT

**Processor Sets/Pinning**

[A] workload can get better performance if each WebSphere Application Server (WAS) instance, a process in itself, is set to run on a separate subset of CPU threads. Keeping a process on a set of CPU threads, and keeping other processes off that set of CPU threads, can improve performance because it preserves CPU cache warmth and NUMA memory locality. In this setup, with 8 WAS instances and 16 cores, each with 4 Simultaneous Multi-Threading (SMT) threads, each WAS instance was pinned to 2 cores, or 8 CPU threads.

The taskset command may be used to assign the CPUs for a program when the program is started. For example:

```sh
taskset -c 0-7 /opt/WAS8.5/profiles/specjprofile1/bin/startServer.sh server1
taskset -c 16-23 /opt/WAS8.5/profiles/specjprofile2/bin/startServer.sh server1
taskset -c 32-39 /opt/WAS8.5/profiles/specjprofile3/bin/startServer.sh server1
taskset -c 48-55 /opt/WAS8.5/profiles/specjprofile4/bin/startServer.sh server1
taskset -c 8-15 /opt/WAS8.5/profiles/specjprofile5/bin/startServer.sh server1
taskset -c 24-31 /opt/WAS8.5/profiles/specjprofile6/bin/startServer.sh server1
taskset -c 40-47 /opt/WAS8.5/profiles/specjprofile7/bin/startServer.sh server1
taskset -c 56-63 /opt/WAS8.5/profiles/specjprofile8/bin/startServer.sh server1
```


**Interrupt Processing**

Usually, the Linux kernel handles network devices by using the so called New API (NAPI), which uses interrupt mitigation techniques, in order to reduce the overhead of context switches: On low traffic network devices everything works as expected, the CPU is interrupted whenever a new packet arrives at the network interface. This gives a low
latency in the processing of arriving packets, but also introduces some overhead, because the CPU has to switch its context to process the interrupt handler. Therefore, if a certain amount of packets per second arrives at a specific network device, the NAPI switches to polling mode for that high traffic device. In polling mode the interrupts are disabled and the network stack polls the device in regular intervals. It can be expected that new packets arrive between two polls on a high traffic network interface. Thus, polling for new data is more efficient than having the CPU interrupted and switching its context on every arriving packet. Polling a network device does not provide the lowest packet processing latency, though, but is throughput optimized and runs with a foreseeable and uniform work load.


**IRQ Pinning**

When processes are pinned to specific sets of CPUs, it can help to pin any interrupts that are used exclusively (or mostly) by those processes to the same set of CPUs. In this setup, each WAS instance was configured with its own IP address. The IP address was configured on a specific Ethernet device. The Ethernet device was handled by one or more interrupts or IRQs. Pinning the IRQs for an Ethernet device to the same set or subset of CPUs of the WebSphere Application Server (WAS) instance that has its IP address on that Ethernet device can help performance.

When you pin IRQs to CPUs, you must keep the `irqbalance` service from setting the CPUs for those IRQs. The `irqbalance` daemon periodically assigns the IRQs to different CPUs depending on the current system usage. It is useful for many system workloads, but if you leave `irqbalance` running it can undo your IRQ CPU pinnings. The heavy-handed approach is to simply turn off the `irqbalance` service and keep it from starting on boot up.

```
# service irqbalance stop
# chkconfig irqbalance off
```

If you need the `irqbalance` service to continue to balance the IRQs that you don't pin, then you can configure `irqbalance` not to change the CPU pinnings for IRQs you pinned. In the `/etc/sysconfig/irqbalance` file, set the `IRQBALANCE_ARGS` parameter to ban `irqbalance` from changing the CPU pinnings for your IRQs.

```
IRQBALANCE_ARGS="--banirq=34 --banirq=35 --banirq=36 --banirq=37 --banirq=38 --banirq=39 --banirq=40 --banirq=41"
```

You must restart the `irqbalance` service for the changes to take effect.

```
# service irqbalance restart
```

To pin the IRQs for an Ethernet device to a CPU or set of CPUs, first you need to find the IRQ numbers the Ethernet device is using. They can be found in the `/proc/interrupts` file.
• The first column in the file lists the IRQs currently being used by the system, each IRQ has its own row.
• The following columns, one for each CPU in the system, list how many times the IRQ was handled on a specific CPU. In the example below, the columns for CPUs beyond CPU1 have been deleted. The file gets very wide when the system has a lot of CPUs.
• The last column lists the name of the IRQ.

In the example that follows, you can see that Ethernet device eth0 has IRQs 34, 35, 36, and 37, and eth1 has IRQs 38, 39, 40, and 41. It is best to read the rows from right to left. Find the device name in the last column, then look at the beginning of the row to determine the assigned IRQ.

<table>
<thead>
<tr>
<th>CPU0</th>
<th>CPU1</th>
<th>&lt;additional CPU columns deleted&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>16:</td>
<td>3546</td>
<td>16486</td>
</tr>
<tr>
<td>29:</td>
<td>17452</td>
<td>0</td>
</tr>
<tr>
<td>30:</td>
<td>4303</td>
<td>0</td>
</tr>
<tr>
<td>31:</td>
<td>133</td>
<td>0</td>
</tr>
<tr>
<td>32:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>33:</td>
<td>417366</td>
<td>0</td>
</tr>
<tr>
<td>34:</td>
<td>8568860</td>
<td>0</td>
</tr>
<tr>
<td>35:</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>36:</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>37:</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>38:</td>
<td>109</td>
<td>0</td>
</tr>
<tr>
<td>39:</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>40:</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>41:</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The CPUs an IRQ is allowed to run on are in the `/proc/irq/<irq-number>/smp_affinity` file. The file contains a hexadecimal bit-mask of the CPUs on which the IRQ is allowed to run. The low order bit is CPU 0. Some Linux distributions also have a `/proc/irq/<irq-number>/smp_affinity_list` file that has the CPU list in human readable form. These files are writable; you can set the CPUs an IRQ is allowed to run on by writing a new value to the file.

Now, let’s say that the first WAS instance is pinned to CPUs 0-3 and that its IP address is on eth0, and that the second WAS instance is pinned to CPUs 4-7 and that its IP address is on eth1. You could pin each of the four IRQs for eth0 to each of the four CPUs to which the first WAS instance is bound, and pin each of the four IRQs for eth1 to each of the four CPUs to which the second WAS instance is bound.

To specify the CPU numbers with a hexadecimal bit-mask, you would write to the `smp_affinity` file.

```bash
# echo 00000001 > /proc/irq/34/smp_affinity
# echo 00000002 > /proc/irq/35/smp_affinity
# echo 00000004 > /proc/irq/36/smp_affinity
# echo 00000008 > /proc/irq/37/smp_affinity
# echo 00000010 > /proc/irq/38/smp_affinity
```
Alternatively, to specify the CPU numbers in a human readable form, you would write to the `smp_affinity_list` file.

```
# echo 0 > /proc/irq/34/smp_affinity_list
# echo 1 > /proc/irq/35/smp_affinity_list
# echo 2 > /proc/irq/36/smp_affinity_list
# echo 3 > /proc/irq/37/smp_affinity_list
# echo 4 > /proc/irq/38/smp_affinity_list
# echo 5 > /proc/irq/39/smp_affinity_list
# echo 6 > /proc/irq/40/smp_affinity_list
# echo 7 > /proc/irq/41/smp_affinity_list
```

However, research has shown that the performance of the IRQ handling is better on the first SMT thread of a core. It is better to combine IRQs on the first SMT thread than to spread them out over all the SMT threads. The PowerLinux systems were configured with SMT4 enabled. The first SMT thread on a core is therefore any CPU number that is evenly divisible by four. So in this example, what you would instead want to do is pin all the IRQs for eth0 to CPU 0 and pin all the IRQs for eth1 to CPU 4.

```
# echo 00000001 > /proc/irq/34/smp_affinity
# echo 00000001 > /proc/irq/35/smp_affinity
# echo 00000001 > /proc/irq/36/smp_affinity
# echo 00000001 > /proc/irq/37/smp_affinity
# echo 00000010 > /proc/irq/38/smp_affinity
# echo 00000010 > /proc/irq/39/smp_affinity
# echo 00000010 > /proc/irq/40/smp_affinity
# echo 00000010 > /proc/irq/41/smp_affinity
```

Or, if using the `smp_affinity_list` file:

```
# echo 0 > /proc/irq/34/smp_affinity_list
# echo 0 > /proc/irq/35/smp_affinity_list
# echo 0 > /proc/irq/36/smp_affinity_list
# echo 0 > /proc/irq/37/smp_affinity_list
# echo 4 > /proc/irq/38/smp_affinity_list
# echo 4 > /proc/irq/39/smp_affinity_list
# echo 4 > /proc/irq/40/smp_affinity_list
# echo 4 > /proc/irq/41/smp_affinity_list
```


**Interrupt Coalescing**

Most modern network adapters have settings for coalescing interrupts. In interrupt
coalescing, the adapter collects multiple network packets and then delivers the packets to the operating system on a single interrupt. The advantage of interrupt coalescing is that it decreases CPU utilization since the CPU does not have to run the entire interrupt code path for every network packet. The disadvantage of interrupt coalescing is that it can delay the delivery of network packets, which can hurt workloads that depend on low network latency. The SPECjEnterprise workload is not sensitive to network latency. For SPECjEnterprise, it is better to conserve CPU utilization, freeing it up for the applications such as WebSphere and DB2.

On some network adapters the coalescing settings are command line parameters specified when the kernel module for the network adapter is loaded. On the Chelseo and Intel adapters used in this setup, the coalescing settings are changed with the `ethtool` utility. To see the coalescing settings for an Ethernet device run `ethtool` with the `-c` option.

```
# ethtool -c eth2
Coalesce parameters for eth2:
Adaptive RX: off  TX: off
stats-block-usecs: 0
sample-interval: 0
pkt-rate-low: 0
pkt-rate-high: 0
rx-usecs: 3
rx-frames: 0
rx-usecs-irq: 0
rx-frames-irq: 0

tx-usecs: 0
tx-frames: 0
tx-usecs-irq: 0
tx-frames-irq: 0

rx-usecs-low: 0
rx-frame-low: 0
tx-usecs-low: 0
tx-frame-low: 0

rx-usecs-high: 0
rx-frame-high: 0
tx-usecs-high: 0
tx-frame-high: 0
```

Many modern network adapters have adaptive coalescing that analyzes the network frame rate and frame sizes and dynamically sets the coalescing parameters based on the current load. Sometimes the adaptive coalescing doesn't do what is optimal for the current workload and it becomes necessary to manually set the coalescing parameters. Coalescing parameters are set in one of two basic ways. One way is to specify a timeout. The adapter holds network frames until a specified timeout and then delivers all the frames it collected. The second way is to specify a number of frames. The adapter holds network frames until it collects the specified number of frames and then delivers all the frames it collected. A combination of the two is usually used.
To set the coalescing settings for an Ethernet device, use the `-C` option for `ethtool` and specify the settings you want to change and their new values. This workload benefited from setting the receive timeout on the WebSphere server to 200 microseconds, the maximum allowed by the Chelseo driver, and disabling the frame count threshold.

```
ethtool -C eth4 rx-usecs 200 rx-frames 0
ethtool -C eth5 rx-usecs 200 rx-frames 0
ethtool -C eth6 rx-usecs 200 rx-frames 0
ethtool -C eth7 rx-usecs 200 rx-frames 0
```

On the database server, increasing the receive timeout to 100 microseconds was sufficient to gain some efficiency. The database server had plenty of idle CPU time, so it was not necessary to conserve CPU utilization.

```
ethtool -C eth2 rx-usecs 100
ethtool -C eth3 rx-usecs 100
ethtool -C eth4 rx-usecs 100
ethtool -C eth5 rx-usecs 100
```


### Consider Disabling IPv6

The setup used only IPv4 connections; it did not need IPv6 support. It got a small boost to performance by disabling IPv6 support in Linux. With IPv6 disabled, the WAS instances didn't spend extra time trying to work with possible IPv6 connections.

IPv6 support can be disabled in the Linux kernel by adding the following options to the kernel command line in the boot loader configuration.

```
ipv6.disable_ipv6=1 ipv6.disable=1
```

Disabling IPv6 support in the Linux kernel guarantees that no IPv6 code will ever be run as long as the system is booted. That may be too heavy-handed. A lighter touch is to let the kernel boot with IPv6 support and then use the `sysctl` facility to dynamically set a kernel variable to disable IPv6.

```
sysctl -w net.ipv6.conf.all.disable_ipv6=1
```

The example above disables IPv6 on all interfaces. You can optionally disable IPv6 support on specific interfaces.

```
sysctl -w net.ipv6.conf.eth0.disable_ipv6=1
sysctl -w net.ipv6.conf.eth1.disable_ipv6=1
```

Huge Pages

The default page size is 4KB. Large pages on Linux are called huge pages, and they are commonly 2MB or 1GB (depending on the processor). In general, large pages perform better for most non-memory constrained workloads because of fewer and faster CPU translation lookaside buffer (TLB) misses. There are two types of huge pages: the newer transparent huge pages (AnonHugePages in /proc/meminfo) and the older hugetlb (HugePages_Total in /proc/meminfo). In general, transparent huge pages are preferred.

Note that there are some potential negatives to huge pages: "huge page use can increase memory pressure, add latency for minor pages faults, and add overhead when splitting huge pages or coalescing normal sized pages into huge pages" (http://developerblog.redhat.com/2014/03/10/examining-huge-pages-or-transparent-huge-pages-performance/).

Transparent Huge Pages

In recent kernel versions, transparent huge page (THP) support is enabled by default and automatically tries to use huge pages: https://www.kernel.org/doc/Documentation/vm/transhuge.txt. The status of THP can be checked with:

```
$ cat /sys/kernel/mm/transparent_hugepage/enabled
[always] never
```

The number of anonymous huge pages allocated can be found in /proc/meminfo

```
$ grep AnonHuge /proc/meminfo
AnonHugePages:  1417216 kB
```

Transparent huge pages use the khugepaged daemon to periodically defragment memory to make it available for future THP allocations. If this causes problems with high CPU usage, defrag may be disabled, at the cost of potentially lower usage of huge pages:

```
It's also possible to limit defrag efforts in the VM to generate hugepages in case they're not immediately free to madvise regions or to never try to defrag memory and simply fallback to regular pages unless hugepages are immediately available. Clearly if we spend CPU time to defrag memory, we would expect to gain even more by the fact we use hugepages later instead of regular pages. This isn't always guaranteed, but it may be more likely in case the allocation is for a MADV_HUGEPAGE region.

echo always > /sys/kernel/mm/transparent_hugepage/defrag
echo madvise > /sys/kernel/mm/transparent_hugepage/defrag
echo never > /sys/kernel/mm/transparent_hugepage/defrag
```

https://www.kernel.org/doc/Documentation/vm/transhuge.txt

AnonHugePages is a subset of AnonPages.

You can check for transparent huge page usage by process in /proc/PID/smaps and look for AnonHugePages.
Important notes about THP:

[THP] requires no modifications for applications to take advantage of it.

An application may mmap a large region but only touch 1 byte of it, in that case a 2M page might be allocated instead of a 4k page for no good. This is why it's possible to disable hugepages system-wide and to only have them inside MADV_HUGEPAGE madvise regions.

https://www.kernel.org/doc/Documentation/vm/transhuge.txt

The amount of memory dedicated to page tables can be found with grep PageTables /proc/meminfo

If your architecture is NUMA and kernel is >= 2.6.14, the huge pages are per NUMA node and so you can see the total huge pages allocated to a process by adding the "huge" elements across nodes in /proc/PID/numa_maps: http://man7.org/linux/man-pages/man7/numa.7.html

Show huge page layout per NUMA node: cat /sys/devices/system/node/node*/meminfo

**hugetlb**

The older method to use huge pages involves libhugetlbfs and complex administration: https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt. Note:

Pages that are used as huge pages are reserved inside the kernel and cannot be used for other purposes. Huge pages cannot be swapped out under memory pressure.

/proc/meminfo contains information on libhugetlbfs usage:

- **HugePages_Total** is the size of the pool of huge pages.
- **HugePages_Free** is the number of huge pages in the pool that are not yet allocated.
- **HugePages_Rsvd** is short for "reserved," and is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made. Reserved huge pages guarantee that an application will be able to allocate a huge page from the pool of huge pages at fault time.
- **HugePages_Surp** is short for "surplus," and is the number of huge pages in the pool above the value in /proc/sys/vm/nr_hugepages. The maximum number of surplus huge pages is controlled by /proc/sys/vm/nr_overcommit_hugepages.
- **Hugepagesize** is the size of each huge page.

The number of hugetlb pages in use is:

**HugePages_Total - HugePages_Free + HugePages_Rsvd**

For example:

- **HugePages_Total**: 8192
- **HugePages_Free**: 1024
- **HugePages_Rsvd**: 1024
- **HugePages_Surp**: 0
- **Hugepagesize**: 2048 kB
In this example, there are no hugetlb pages in use, although 1GB is reserved by some processes.

More information: [http://lwn.net/Articles/374424/](http://lwn.net/Articles/374424/)

**Process Limits**

Modify the core, data, fsize, nproc, and rss ulimit values to unlimited at least for the user that will run WAS related processes in `/etc/security/limits.conf` ([http://www-01.ibm.com/support/docview.wss?uid=swg21469413](http://www-01.ibm.com/support/docview.wss?uid=swg21469413)). For example:

- Maximum number of open files: `nofile (ulimit -n) = 1048576`
- Maximum number of user processes: `ulimit -u unlimited`
- Maximum stack size: `ulimit -s 524288`

The maximum number of processes and threads is controlled by `/proc/sys/kernel/threads-max`: "This file specifies the system-wide limit on the number of threads (tasks) that can be created on the system." ([http://man7.org/linux/man-pages/man5/proc.5.html](http://man7.org/linux/man-pages/man5/proc.5.html)) Each thread also has a maximum stack size, so virtual and physical memory must support your requirements.

The maximum number of PIDs is controlled by `/proc/sys/kernel/pid_max`: "This file specifies the value at which PIDs wrap around (i.e., the value in this file is one greater than the maximum PID). The default value for this file, 32768, results in the same range of PIDs as on earlier kernels. On 32-bit platforms, 32768 is the maximum value for pid_max. On 64-bit systems, pid_max can be set to any value up to $2^{22}$ (PID_MAX_LIMIT, approximately 4 million)." ([http://man7.org/linux/man-pages/man5/proc.5.html](http://man7.org/linux/man-pages/man5/proc.5.html))

Modify `ulimit -u` (nproc) to 131072. If you have a file named `/etc/security/limits.d/90-nproc.conf`, then edit the * line; otherwise, add soft and hard nproc limits to `/etc/security/limits.conf` ([http://www-01.ibm.com/support/docview.wss?uid=swg21648497](http://www-01.ibm.com/support/docview.wss?uid=swg21648497)).

**Crontab**

Review all users' crontabs and the processing that they do. Some built-in crontab processing such as monitoring and file search may have significant performance impacts.

**Processor Scheduling**


The Linux Completely Fair Scheduler (CFS) first appeared in the 2.6.23 release of the Linux kernel in October 2007. The algorithms used in the CFS provide efficient scheduling for a wide variety of system and workloads. However, for this particular workload there is one behavior of the CFS that cost a few percent of CPU utilization.

In the CFS, a thread that submits I/O, blocks and then is notified of the I/O completion preempts the currently running thread and is run instead. This behavior is great for applications such as video streaming that need to have low latency for handling the I/O, but it can actually hurt SPECjEnterprise performance. In SPECjEnterprise, when a thread submits I/O, such as sending a response out on the network, the I/O thread is in no hurry to
handle the I/O completion. Upon I/O completion, the thread is simply finished with its work. Moreover, when an I/O completion thread preempts the current running thread, it prevents the current thread from making progress. And when it preempts the current thread it can ruin some of the cache warmth that the thread has created. Since there is no immediate need to handle the I/O completion, the current thread should be allowed to run. The I/O completion thread should be scheduled to run just like any other process.

The CFS has a list of scheduling features that can be enabled or disabled. The setting of these features is available through the debugfs file system. One of the features is WAKEUP_PREEMPT. It tells the scheduler that an I/O thread that was woken up should preempt the currently running thread, which is the default behavior as described above. To disable this feature, you set NO_WAKEUP_PREEMPT (not to be confused with NO_WAKEUP_PREEMPTION) in the scheduler's features.

```
mount -t debugfs debugfs /sys/kernel/debug
echo NO_WAKEUP_PREEMPT > /sys/kernel/debug/sched_features
umount /sys/kernel/debug
```

Unfortunately, the NO_WAKEUP_PREEMPT scheduler feature was removed in Linux kernel version 3.2. It is and will be available in the RedHat Enterprise Linux 6 releases. It is not available in the latest SUSE Linux Enterprise Server 11 Service Pack 2. There are some other scheduler settings that can achieve close to the same behavior as NO_WAKEUP_PREEMPT.

You can use the `sched_min_granularity_ns` parameter to disable preemption. `sched_min_granularity_ns` is the number of nanoseconds a process is guaranteed to run before it can be preempted. Setting the parameter to one half of the value of the `sched_latency_ns` parameter effectively disables preemption. `sched_latency_ns` is the period over which CFS tries to fairly schedule all the tasks on the runqueue. All of the tasks on the runqueue are guaranteed to be scheduled once within this period. So, the greatest amount of time a task can be given to run is inversely correlated with the number of tasks; fewer tasks means they each get to run longer. Since the smallest number of tasks needed for one to preempt another is two, setting `sched_min_granularity_ns` to half of `sched_latency_ns` means the second task will not be allowed to preempt the first task.

The scheduling parameters are located in the `/proc/sys/kernel/` directory. Here is some sample bash code for disabling preemption.

```
# LATENCY=$(cat /proc/sys/kernel/sched_latency_ns)
# echo $((LATENCY/2)) > /proc/sys/kernel/sched_min_granularity_ns
```

The parameter `sched_wakeup_granularity_ns` is similar to the `sched_min_granularity_ns` parameter. The documentation is a little fuzzy on how this parameter actually works. It controls the ability of tasks being woken to preempt the current task. The smaller the value, the easier it is for the task to force the preemption. Setting `sched_wakeup_granularity_ns` to one half of `sched_latency_ns` can
also help alleviate the scheduling preemption problem.


**IBM Java on Linux**

"Java applications that use synchronization extensively might perform poorly on Linux distributions that include the Completely Fair Scheduler. The Completely Fair Scheduler (CFS) is a scheduler that was adopted into the mainline Linux kernel as of release 2.6.23. The CFS algorithm is different from the scheduling algorithms for previous Linux releases. It might change the performance properties of some applications. In particular, CFS implements sched_yield() differently, making it more likely that a yielding thread is given CPU time regardless.

If you encounter this problem, you might observe high CPU usage by your Java application, and slow progress through synchronized blocks. The application might appear to stop because of the slow progress.

There are two possible workarounds:

1. Invoke the JVM with the additional argument -Xthr:minimizeUserCPU.
2. Configure the Linux kernel to use a heuristic for sched_yield() that is more compatible with earlier versions, by setting the sched_compat_yield tunable kernel property to 1. For example:

   ```
   echo "1" > /proc/sys/kernel/sched_compat_yield
   ```

Do not use these workarounds unless you are experiencing poor performance.

"You can test whether these optimizations are negatively affecting your application by running the following test. Use the following command-line option to revert to behavior that is closer to earlier versions and monitor application performance: -Xthr:noCfsYield" (https://www.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.aix.70.doc/diag/problem_determination/optimizations_pd.html)

**RedHat Enterprise Linux (RHEL)**

**Tuning Profiles**

"The Tuned package is a tuning profile delivery mechanism shipped in Red Hat Enterprise Linux 6 and 7. It is the primary vehicle in which research conducted by Red Hat's Performance Engineering Group is provided to customers. [...] The network-latency tuned profile (new in Red Hat Enterprise Linux 7) attempts to address the most common causes of latency introduced by power management by disabling deeper c-states and requesting the maximum performance from the Intel CPU P-state driver. However, depending on the BIOS configuration, settings applied by tuned may be overridden or not applied."
Consider testing with RedHat's "performance" or "network-latency" system profiles:

```
sudo tuned-adm profile latency-performance
sudo tuned-adm profile networky-latency
sudo tuned-adm profile networky-throughput
```

Current profile:

```
sudo tuned-adm list
```

Profile details in `/usr/lib/tuned/<profile>/tuned.conf`

See more:

- [https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v2.1.pdf](https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v2.1.pdf)
- [https://access.redhat.com/articles/1323793](https://access.redhat.com/articles/1323793)

**Automatic Bug Reporting Tool (ABRT)**


```
$ cat /proc/sys/kernel/core_pattern
|/usr/libexec/abrt-hook-ccpp %s %c %p %u %g %t e
```

ABRT may decide to copy or place the core file into a subdirectory of `/var/spool/abrt`. Ensure there is sufficient space. Any warnings or errors in ABRT processing will be found in `/var/log/messages`. Stopping the ABRT services will revert the core_pattern.

**Other Tips**

- Print kernel boot parameters: `cat /proc/cmdline`
- Change kernel log level: `echo 5 > /proc/sys/kernel/printk`

**Linux on Power**


Consider testing with -Xnodfpbd because "The hardware instructions can be slow."

Hardware Prefetching

Consider disabling hardware prefetching because Java does it in software. "[Use] the ppc64_cpu utility (available in the powerpc-utils package) to set the pre-fetch depth to 1 (none) in the DSCR."

```
# ppc64_cpu --dscr=1
```

Idle Power Saver

Idle Power Saver, [which is enabled by default], will put the processor into a power saving mode when it detects that utilization has gone below a certain threshold for a specified amount of time. Switching the processor into and out of power saving mode takes time. For sustained peak performance it is best not to let the system drop into power saving mode. Idle Power Saver can be disabled by using the web interface to the Advanced System Management Interface (ASMI) console. Navigate to System Configuration -> Power Management -> Idle Power Saver. Set the Idle Power Saver value to Disabled, then click on the "Save settings" button on the bottom of the page.


Adaptive Frequency Boost

The Adaptive Frequency Boost feature allows the system to increase the clock speed for the processors beyond their nominal speed as long as environmental conditions allow it, for example, the processor temperature is not too high. Adaptive Frequency Boost is enabled by default. The setting can be verified (or enabled if it is disabled) by using the web interface to the Advanced System Management Interface (ASMI) console. Navigate to Performance Setup -> Adaptive Frequency Boost. Change the setting to Enabled, then click on the "Save settings" button.

Dynamic Power Saver (Favor Performance) Mode

The PowerLinux systems have a feature called Dynamic Power Saver that will dynamically adjust the processor frequencies to save energy based on the current processor utilization. The Dynamic Power Saver mode can be set to favor performance by using the web interface to the ASMI console. Navigate to System Configuration -> Power Management -> Power Mode Setup. Select Enable Dynamic Power Saver (favor performance) mode, then click on the "Continue" button.


64-bit DMA Adapter Slots for Network Adapters

The 64-bit direct memory access (DMA) adapter slots are a feature on the newer IBM POWER7+ systems. 64-bit DMA enables a faster data transfer between I/O cards and the system by using a larger DMA window, possibly covering all memory. On the PowerLinux 7R2 system, two of the adapter slots, slots 2 and 5, are enabled with 64-bit DMA support. On each system the two network cards were installed in the two 64-bit DMA slots. Using the 64-bit DMA slots resulted in a noticeable improvement in network performance and CPU utilization.


Scaling Up or Out

One question for tuning a multi-threaded workload for increased capacity is whether to scale up by adding more processor cores to an instance of an application or to scale out by increasing the number of application instances, keeping the number of processor cores per application instance the same.

The performance analysis for this workload on the Power architecture has shown that the WebSphere Application Server (WAS) performs best with two processor cores and their attending SMT threads. Therefore, when increasing the capacity of a POWER system running WAS it is best to increase the number of WAS instances, giving each instance two processor cores. The WAS setup for SPECjEnterprise2010 ran eight WAS instances.

...[If] the WAS instances have to listen on the same port... By default, a WAS instance is configured with multi-home enabled, which means it listens for requests on its port on all of
the IP addresses on the system. If multiple WAS instances are running, they cannot all be
allowed to listen for requests on all the IP addresses. They would end up stepping on each
other and would not function correctly. If multiple WAS instances are running, multi-home
must be disabled and each WAS instance must be configured to listen on a different IP
address. For instructions on how to configure an application server to use a single network
interface, see Configuring an application server to use a single network interface [4] in the
WebSphere Application Server Version 8.5 Information Center.

Since a system cannot have multiple IP addresses on the same subnet, the IP address of
each WAS instance must be on its own Ethernet device. This can easily be done if the
number of Ethernet devices on the system is greater than or equal to the number of WAS
instances, the IP addresses for the WAS instances can each be put on their own Ethernet
device.

If the system has fewer Ethernet devices than the number of WAS instances, then aliases
can be used to create multiple virtual devices on a single physical Ethernet device. See
Guide for details on how to configure an alias interface.

https://www.ibm.com/developerworks/community/wikis/home?
lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/SPECjEnterprise2010%20-%20A%20performance%20case%20study

Linux on System z (zLinux, s390)

In general, we recommend setting QUICKDSP on for production guests and server virtual
machines that perform critical system functions.

You can get a sense of the system your Linux virtual server is running on by issuing cat
/proc/sysinfo


See value of WAS on zLinux here: http://www-
03.ibm.com/support/techdocs/atmsastr.nsf/5cb5ed706d254a8186256c71006d2e0a/981aad32cbab471886257604001e1ec8/$FILE/WP101532%20-%20Why%20WAS%20zOS%20-%202013%20Version%20-%20CHARTS.pdf

The zLinux "architecture" is sometimes referred to as s390.

z/VM has three storage areas: central store (cstore), expanded store (xstore), and page volumes. The
first two are RAM and the last is disk.

Discontiguous Saved Segments (DCSS)

Discontiguous Saved Segments (DCSS) may be mounted in zLinux to share data across guests, thus
potentially reducing physical memory usage: http://www.vm.ibm.com/linux/dcss/dcsslarg.pdf. DCSS can also be used as an in-memory filesystem.

AIX


AIX Recipe

1. **CPU core(s)** should not be consistently saturated.
2. Generally, physical memory should never be saturated and the operating system should not page memory out to disk.
3. Input/Output interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. TCP/IP and network tuning, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.
9. Bind your processes properly based on system topology.
10. Use MCM memory affinity where appropriate.
11. Find the optimal SMT configuration for the machine.
12. Find the optimal hardware prefetching setting for your workload.
13. Apply recommended tuning for Java applications.
14. For large multi-threaded apps, use profiling to make sure that work is allocated equally amongst threads.
15. For apps that use a lot of network I/O, tune networking parameters.
16. For apps that make heavy use of native memory, experiment with and use the optimal malloc algorithm.
17. Use profiling to evaluate the effects of tuning other parameters.

Also review the general topics in the **Operating Systems chapter**.

General


- **AIXTHREAD_SCOPE=S**
  - The default value for this variable is P, which signifies processwide contention scope (M:N). S signifies systemwide contention scope (1:1). For Java applications, you should set this value to S.
- **AIXTHREAD_MUTEX_DEBUG=OFF**
• Maintains a list of active mutexes for use by the debugger.

• AIXTHREAD_COND_DEBUG=OFF
  • Maintains a list of condition variables for use by the debugger.

• AIXTHREAD_RWLOCK_DEBUG=OFF
  • Maintains a list of active mutual exclusion locks, condition variables, and read-write locks for use by the debugger. When a lock is initialized, it is added to the list if it is not there already. This list is implemented as a linked list, so searching it to determine if a lock is present or not has a performance implication when the list gets large. The problem is compounded by the fact that the list is protected by a lock, which is held for the duration of the search operation. Other calls to the pthread_mutex_init() subroutine must wait while the search is completed. For optimal performance, you should set the value of this thread-debug option to OFF. Their default is ON.

• SPINLOOPTIME=500
  • Number of times that a process can spin on a busy lock before blocking. This value is set to 40 by default. If the tprof command output indicates high CPU usage for the check_lock routine, and if locks are usually available within a short amount of time, you should increase the spin time by setting the value to 500 or higher. Note that this may not affect JVM monitors, which implement their own tiered locking scheme and only revert to OS locking primitives when other alternatives fail.

These can be set for WAS processes by adding "environment entries" for each JVM: [http://www-01.ibm.com/support/docview.wss?uid=swg21254153](http://www-01.ibm.com/support/docview.wss?uid=swg21254153)

"For filesystems with a high rate of file access, performance can be improved by disabling the update of the access time stamp. This option can be added to a filesystem by using the ":-o noatime" mount option, or permanently set using "chfs -a options=noatime." ([http://www-01.ibm.com/support/docview.wss?uid=isg3T1012054](http://www-01.ibm.com/support/docview.wss?uid=isg3T1012054))

Note, if IBM Java uses shmget to allocate the Java heap, then it will immediately mark the shared memory region for deletion, so that if the JVM crashes, the memory will be released. Therefore, if you find large shared memory regions marked for deletion (ipcs -Smqsa -1 | egrep "^m" | egrep " D"), this is most likely the reason.

Query operating system version: oslevel

### Central Processing Unit (CPU)

Query physical processor information:

```bash
# prtconf
System Model: IBM,9119-FHB
Processor Type: PowerPC POWER7
Number Of Processors: 2
Processor Clock Speed: 4004 MHz...
```

### Simultaneous Multithreading (SMT)

The smtctl command may be used to query and change CPUs' SMT mode ([http://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.cmds5/smtctl.htm](http://www.ibm.com/support/knowledgecenter/ssw_aix_72/com.ibm.aix.cmds5/smtctl.htm)):

```
# smtctl
```
This system supports up to 4 SMT threads per processor.
SMT is currently enabled...
proc0 has 4 SMT threads...

It is important to use the most optimal SMT setting for the machine, based on the number of CPU-intensive processes running on the machine and their threading characteristics. If the machine is running one or a few single threaded applications then disabling SMT may be the most optimal setting. On the other hand, if the machine is running a large, multi-threaded application or several CPU-intensive processes, running in SMT4 mode may be the most optimal setting.

If the workload is very CPU intensive, consider a lower SMT value, and if the workload is very memory/data/IO intensive, consider a higher SMT value.

CPU Terminology

See the discussion of CPU core(s) as background.

- Physical Processor: An IBM Power CPU core.
- Virtual Processor: The logical equivalent of a Physical Processor, although the underlying Physical Processor may change over time for a given Virtual Processor.
- Logical Processor: If SMT is disabled, a Virtual Processor. If SMT is enabled, an SMT thread in the Virtual Processor.

Micro-Partitioning

The LPAR always sees the number of CPUs as reported by "Online Virtual CPUs" in lparstat -i:

```
# lparstat -i
Type : Shared-SMT-4
Mode : Uncapped
Entitled Capacity : 0.20
Online Virtual CPUs : 2 ...
```

IBM generally recommends (Virtual CPUs) / (Physical CPUs) <= 3 for Power7, ideally 1-2. Also note that a virtual processor may be a CPU core thread rather than a CPU core. Review the Operating Systems chapter for background on CPU allocation.

These CPUs may further be hyperthreaded depending on the SMT mode.

If the LPAR is capped, it can only use up to its entitlement, spread across the online virtual CPUs. In general, if using capped LPARs, it's recommended to set entitlement equal to online virtual CPUs. If the LPAR is uncapped, it can use up to all of the online virtual CPUs, if available.

The benefit of Micro-Partitioning is that it allows for increased overall utilization of system resources by applying only the required amount of processor resource needed by each partition. But due to the overhead associated with maintaining online virtual processors, consider the capacity requirements when choosing values for the attributes.

For optimal performance, ensure that you create the minimal amount of partitions, which decreases the overhead of scheduling virtual processors.

CPU-intensive applications, like high performance computing applications, might not be

Page 97
suitable for a Micro-Partitioning environment. If an application uses most of its entitled
processing capacity during execution, you should use a dedicated processor partition to
handle the demands of the application.

Even if using uncapped, entitled capacity should not exceed 100% because the lack of processor
affinity may cause performance problems: http://www-01.ibm.com/support/docview.wss?
uid=isg3T1024788

For PowerVM, a dedicated partition is preferred over a shared partition or a workload partition for the
system under test.

By default, CPU folding occurs in both capped and uncapped modes, with the purpose being to
increase CPU cache hits
mngmnt_part.htm). In general, CPU folding should not be disabled, but low values of CPU folding (as
seen in nmon, for example) may indicate low entitlement. Processor folding may be disabled with:

```
# schedo -o vpm_xvcpus=-1
```

Use mpsstat to review processor affinity:
https://www.ibm.com/developerworks/mydeveloperworks/blogs/aixpert/entry/mpstat_d_and_the_undo
cumented_stats133?lang=en

**vmstat**

Query processor usage

```
$ vmstat -tw 30 2
System configuration: lcpu=8 mem=8192MB ent=0.20
kthr memory page
faults cpu time
------- --------------------- ------------------------------------
------------------ ----------------------- --------
r   b        avm        fre    re    pi    po    fr     sr    cy    in
sy    cs us sy id wa pc    ec hr mi se
9   0     934618     485931     0     0     0     0      0     0    18
2497 1299  4 12.84  0  0.06  27.5 11:49:44
6   0     934629     485919     0     0     0     0      0     0    21
13938 3162  56 11.32  0  0.29 142.9 11:50:14
```

Key things to look at:

- The "System configuration" line will report the number of logical CPUs (in this example, 8),
  which may be more than the number of physical CPUs (due to SMT).
- **r**: This is the run queue which is the sum of the number of threads currently running on the
  CPUs plus the number of threads waiting to run on the CPUs. This number should rarely go
  above the number of logical CPUs.
- **b**: This is the number of threads which are blocked, usually waiting for I/O, and should usually
  be zero.
- **pi/po**: Pages in and pages out, respectively, should usually be zero (pi in particular).
- **us/sy/id/wa**: These report the processor usage in different dimensions.
- **pc**: This reports the processor usage as a fraction of the number of physical CPUs.
- **ec**: This reports the processor usage as a fraction of the number of entitled CPUs.

### topas

Topas Monitor for host: aix4prt

<table>
<thead>
<tr>
<th>EVENTS/QUEUES</th>
<th>EVENTS/QUEUES</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mon Apr 16 16:16:50 2001</strong></td>
<td><strong>Interval: 2</strong></td>
</tr>
<tr>
<td>Readch</td>
<td>4864</td>
</tr>
<tr>
<td>Writech</td>
<td>34280</td>
</tr>
<tr>
<td>Kernel</td>
<td>63.1</td>
</tr>
<tr>
<td>Rawin</td>
<td>0</td>
</tr>
<tr>
<td>User</td>
<td>36.8</td>
</tr>
<tr>
<td>Ttyout</td>
<td>0</td>
</tr>
<tr>
<td>Wait</td>
<td>0.0</td>
</tr>
<tr>
<td>Igets</td>
<td>0</td>
</tr>
<tr>
<td>Idle</td>
<td>0.0</td>
</tr>
<tr>
<td>Namei</td>
<td>4</td>
</tr>
<tr>
<td>Cswitch</td>
<td>5984</td>
</tr>
<tr>
<td>Syscall</td>
<td>15776</td>
</tr>
</tbody>
</table>

### MEMORY

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Real,MB</strong></td>
<td><strong>1023</strong></td>
</tr>
<tr>
<td><strong>Comp</strong></td>
<td><strong>27.0</strong></td>
</tr>
<tr>
<td><strong>HDisk</strong></td>
<td><strong>0.0</strong></td>
</tr>
<tr>
<td><strong>Noncomp</strong></td>
<td><strong>73.9</strong></td>
</tr>
<tr>
<td><strong>Client</strong></td>
<td><strong>0.5</strong></td>
</tr>
<tr>
<td><strong>Name</strong></td>
<td><strong>java</strong></td>
</tr>
<tr>
<td><strong>PID</strong></td>
<td><strong>16684</strong></td>
</tr>
<tr>
<td><strong>CPU%</strong></td>
<td><strong>83.6</strong></td>
</tr>
<tr>
<td><strong>PgsP</strong></td>
<td><strong>35.1</strong></td>
</tr>
</tbody>
</table>

### NETWORK

<table>
<thead>
<tr>
<th>NETWORK</th>
<th>NETWORK</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>lo0</strong></td>
<td><strong>213.9</strong></td>
</tr>
<tr>
<td><strong>tr0</strong></td>
<td><strong>34.7</strong></td>
</tr>
<tr>
<td><strong>Network KBPS</strong></td>
<td><strong>I-Pack</strong></td>
</tr>
<tr>
<td><strong>O-Pack</strong></td>
<td><strong>KB-In</strong></td>
</tr>
<tr>
<td><strong>KB-Out</strong></td>
<td><strong>Steals</strong></td>
</tr>
<tr>
<td><strong>Waitqueue</strong></td>
<td><strong>1580</strong></td>
</tr>
</tbody>
</table>

### PAGE SPACE

<table>
<thead>
<tr>
<th>PAGE SPACE</th>
<th>PAGE SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size,MB</strong></td>
<td><strong>512</strong></td>
</tr>
<tr>
<td><strong>Used</strong></td>
<td><strong>1.2</strong></td>
</tr>
<tr>
<td><strong>Free</strong></td>
<td><strong>98.7</strong></td>
</tr>
</tbody>
</table>

The table in the bottom left corner shows the Name, process ID (PID), and CPU usage (CPU%) of processes using the most CPU. If the available CPU is 100% utilized, the Java process is being limited. In the case above, two Java processes are causing 100% CPU utilization, so CPU contention is occurring between them.
**CPU Utilization Reporting Tool (curt)**

The curt tool uses kernel trace data into exact CPU utilization for a period of time:  

Generate curt input data:  

Generate curt output:  

**nmon**

nmon is similar to topas in that it has dynamically updating graphical views of various system statistics. It used to be a standalone utility but it has recently been integrated into the operating system:  

It is generally recommended to always run nmon (or something similar) in the background.

Start nmon for essentially unlimited collection with a 60 second interval:

```
# su
# cd /var/tmp/
# nohup nmon -fT -s 60 -c 1000000 && sleep 1 && cat nohup.out
```

Executing this command will start the nmon collector in the background, so explicitly putting it into the background (using `&`) is not necessary. This will create a file with the name $HOST_$STARTDAY_$STARTTIME.nmon

Note that any errors starting nmon (such as inadequate file permissions when trying to write to the specified directory) will go to nohup.out, so it is important to check nohup.out to make sure it started correctly. You can also run `ps -elfx | grep nmon` to make sure it started.

When you want to stop nmon, run:

```
# su
# ps -elf | grep nmon | grep -v grep | awk '{print $4}' | xargs kill -USR2
```

We recommend using the Java based nmon visualizer which can be found here:  
The older spreadsheet based visualizer can be found here:

**tprof**

The tprof command, an AIX operating system provided utility, uses sampling at the processor level to determine which process, thread, and routine are using CPU time. tprof periodically samples which thread and instructions are being run on each processor and maps this information back to the process and routine. The information is not 100% accurate, but is representative and gathered with very little performance overhead.

The output from the tprof -skex sleep 60 command produces a file called sleep.prof [after 60 seconds of system-wide profiling]:

<table>
<thead>
<tr>
<th>Process</th>
<th>FREQ</th>
<th>Total Kernel</th>
<th>User</th>
<th>Shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>Other</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wait</td>
<td>8</td>
<td>30387</td>
<td>30387</td>
<td>0</td>
</tr>
<tr>
<td>java</td>
<td>34</td>
<td>17533</td>
<td>9794</td>
<td>0</td>
</tr>
<tr>
<td>/usr/sbin/syncd</td>
<td>2</td>
<td>91</td>
<td>91</td>
<td>0</td>
</tr>
<tr>
<td>/usr/bin/tprof</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
This section covers the CPU usage as a whole across all the CPU using processes. (... Kernel: The subset of the total samples that were in Kernel routines. User: The subset of the total samples that were in User routines. Shared: The subset of the total samples that were in Shared library routines. For a Java application, this value represents time spent inside the Java runtime itself or time spent running JNI code. Other: The subset of the total samples that were in Shared library routines. For a Java application, this value represents time spent running the Java methods.)

The value in the Total column for the Java executable compared to the overall Total shows the percentage of overall CPU being used by the Java processes. In this case, 
\((17533/48023) \times 100 = 36.5\%\). The values of Kernel, Shared, and Other for the Java executable shows how much time was spent in the Kernel, running Java runtime support routines, and how much time was spent running the Java methods themselves.

Old Java Diagnostic Guide

By default, tprof does not provide method names for Java user code samples (seen as hexadecimal addresses in SEGMENT-N sections). AIX ships with a JVMTI agent (libjpa) that allows tprof to see method names; however, if you’ve isolated the processor usage in tprof to user Java code, then it is generally better to use a Java profiler such as Health Center. To use the agent pass -agentlib:jpa to a 32-bit JVM or -agentlib:jpa64 to a 64-bit JVM.

Per-thread CPU usage

The next section of interest is the per-process and per-thread breakdown:

<table>
<thead>
<tr>
<th>Process</th>
<th>PID</th>
<th>TID</th>
<th>Total Kernel</th>
<th>User</th>
<th>Shared</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait</td>
<td>53274</td>
<td>61471</td>
<td>4262</td>
<td>4262</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>61470</td>
<td>69667</td>
<td>3215</td>
<td>3215</td>
<td>0</td>
</tr>
<tr>
<td>java</td>
<td>413760</td>
<td>872459</td>
<td>1208</td>
<td>545</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>413760</td>
<td>925875</td>
<td>964</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>413760</td>
<td>790723</td>
<td>759</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>413760</td>
<td>1020037</td>
<td>757</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>
This section shows the same breakdown, but shows it on an individual thread basis, providing the Process ID (PID) and Thread ID (TID) for each thread. [The Total column represents the CPU usage for each Java thread, identified by a thread identifier (TID).] In this example... the [CPU] spread is mostly even across the threads in the process (the Total value is similar for all of the java TIDs). This information implies that the performance of the application is being limited by points of contention or delay in the Java process, which is preventing it from scaling to use all of the available CPU. If a deadlock was present, the CPU usage by the Java process would be low or zero. If threads were looping, the Java CPU usage would be approaching 100% [and a] small subset of the threads would account for all of that CPU time.

For threads of interest, note the TID values. You can convert these values to a hexadecimal value and look up the threads in [a] javacore.txt file to discover if the thread is part of a thread pool, and to gain an understanding of the kind of work that the thread does, using the thread stack trace in the javacore.txt file. For example, the TID of 1020037 is F9085 in hexadecimal and maps to the "native ID" value in the javacore.txt file.

Old Java Diagnostic Guide

**perfpmr.sh**


The default parameters usually run for about 30 minutes:

```
# perfpmr.sh 600
```

The number of seconds passed (in the above example, 600) is not the duration for the entire script, but the maximum for parts of it (e.g. tcpdump, filemon, etc.). For the minimum duration of 60 seconds, the total duration will be about 10 minutes.

One interesting thing to do is process the system trace:

```
# perfpmr.sh -x trace.sh -r
```

This creates a file trace.int. Find all file system system calls:

```
$ grep java trace.int | grep lookuppn
```
If you see a lot of activity to the /dev/null device, e.g.:

```
107 -6947396- 64 14288867 2.183578 lookuppn exit: '/dev/null' =
vnode F1000A03000D1130
```

Though this is to the bit bucket, it will cause the inode for the /dev/null device to be updated about access times and modification times. To make this more efficient, run the following dynamic command:

```
# raso -p -o devnull_lazytime=1
```

To search for processor affinity statistics, review curt.out:

```
# curt -i trace.tr -n trace.syms -est -r PURR -o curt.out
```

### truss

Truss traces system calls:

```
$ truss -d -i -s\!all -o truss.out -p $PID
```

### Physical Memory (RAM)

Query paging spaces:

```
# lsps -a
```

<table>
<thead>
<tr>
<th>Page Space</th>
<th>Physical Volume</th>
<th>Volume Group</th>
<th>Size %Used</th>
<th>Active</th>
<th>Auto</th>
</tr>
</thead>
<tbody>
<tr>
<td>hd6</td>
<td>hdisk0</td>
<td>rootvg</td>
<td>1024MB</td>
<td>2</td>
<td>yes</td>
</tr>
<tr>
<td>lv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### vmstat

When the physical memory is full, paging (also known as swapping) occurs to provide additional memory. Paging consists of writing the contents of physical memory to disk, making the physical memory available for use by applications. The least recently used information is moved first. Paging is expensive in terms of performance because, when the required information is stored on disk it must be loaded back into physical memory, which is a slow process.

Where paging occurs, Java applications are impacted because of garbage collection. Garbage collection requires every part of the Java heap to be read. If any of the Java heap has been paged out, it must be paged back when garbage collection runs, slowing down the garbage collection process.

The vmstat output shows whether paging was taking place when the problem occurred. vmstat output has the following format:

```
kthr memory page faults cpu time
```

```
0 0 45483 221 0 0 0 0 1 0 224 326 362 24 7 69 0 15:10:22
0 0 45483 220 0 0 0 0 0 0 159 83 53 1 1 98 0 15:10:23
2 0 45483 220 0 0 0 0 0 0 145 115 46 0 9 90 1 15:10:24
```
The columns of interest are pi and po (page in and page out) for AIX... Nonzero values indicate that paging is taking place.

Old Java Diagnostic Guide

**svmon**

Unless otherwise noted, svmon numbers, such as inuse and virtual, are in numbers of frames, which are always 4KB each, even if there are differently sized pages involved.

svmon can be used to look at the total memory usage of all the processes on the machine:

```bash
$ svmon -G

size       inuse       free        pin    virtual
memory      524288     297790     226498      63497     107144
pg space   131072        257

work       pers       clnt
pin           63497          0          0
in use       107144     164988      25658
```

The values in the svmon -G output have the following meanings:

- **memory**: pages of physical memory (RAM) in the system
- **pg space**: pages of paging space (swap space) in the system
- **pin**: pages which can only be stored in physical memory and may not be paged to disk
- **in use**: pages which are currently backed by physical memory

**Columns**

- **size**: the total size of the resource
- **inuse**: the number of pages which are currently being used
- **free**: the number of pages which are currently not being used
- **pin**: the number of pages which are currently in use that can only be stored in physical memory
- **virtual**: the number of pages that have been allocated in the process virtual space
- **work**: the number of pages being used for application data
- **pers**: the number of pages being used to cache local files
- **clnt**: the number of pages being used to cache NFS files

Memory inuse on the first row is the physical memory being used. This is split on the second row between work for processes, pers for file cache and clnt for NFS file cache.

If the memory inuse value is equal to the memory size value, then all the physical memory is being used. Some of this memory will most likely be used to cache file systems as the AIX kernel allows file caching to use up to 80% of the physical memory by default. Whilst file caching should be released before paging out application data, depending on system demand the application memory pages may be swapped out. This maximum usage of the
physical memory by file caching can be configured using the AIX vmtune command along with the the minperm and maxperm values. In addition, it is recommended that you set strict_maxperm to 1 in order to prevent AIX from overriding the maxperm setting.

If all the physical memory is being used, and all or the majority of the in use memory shown on the fourth row is for work pages, then the amount of physical memory should be increased. [It is suggested that the rate of increase be similar to the amount of paging space used (see pg space inuse value).]

Old Java Diagnostic Guide

32-bit processes have 16 segments. 64-bit processes have up to $2^{36}$ segments. A segment is always 256MB.

Physical memory pages are called memory frames.

Dynamic page promotion occurs when a set of contiguous pages (e.g. 4K) add up to a page of the next higher size (e.g. 16 4K pages = one 64K page). This is done by psmd (Page Size Management Daemon).

Larger page sizes may reduce page faults and are more efficient for addressing, but may increase overall process size due to memory holes.

The VSID is a system-wide segment ID. If two processes are referencing the same VSID, then they are sharing the same memory.

The ESID (effective segment ID) is a process level segment ID. A typical virtual address, e.g. 0xF1000600035A6C00 starts with the segment and the last 7 hex digits are the page/offset.

Segment types:
- **Working**
  - Kernel: In kernel space and addressable by all processes
  - Text/shared library text: Executable code. Normally shared by instances of the same process.
  - Data: Private data segment
  - mbuf: Network memory
- **Mmap/shmat'ed**: Mapped by 0 or more processes.
- **Persistent**: JFS file segment
- **Client**: Non-JFS file segment (e.g. JFS2/NFS/Veritas/etc)

Inuse: Number of memory frames actually in use.

Virtual: Number of virtual pages.

Pin: Pages that cannot be stolen by lrud.

When a segment has multiple page sizes, the virtual/inuse/pin/pgsp columns display in 4k units.

Account for all pages on the system: svmon -G

File cache size can be determined by adding pers and clnt inuse values.

svmon virtual value or vmstat avm (active virtual memory) column represents the maximum memory in 4K pages needed for programs (non-filesystem cache).
32-bit Memory Model

The 32-bit AIX virtual memory space is split into 16, 256MB segments (0x0 – 0x15). Segment 0x0 is always reserved for the kernel. Segment 0x1 is always reserved for the executable code (java). The rest of the segments may be laid out in different ways depending on the LDR_CNTRL=MAXDATA environment variable or the maxdata parameter compiled in the executable (http://www.ibm.com/developerworks/library/j-nativememory-aix/)

By default, IBM Java will choose a generally appropriate MAXDATA value depending on -Xmx (http://www.ibm.com/support/knowledgecenter/SSYKE2_7.1.0/com.ibm.java.aix.71.doc/user/aix_auto_ldr_cntrl.html):

- Xmx > 3GB: MAXDATA=0@DSA = 3.5GB user space, 256MB malloc, 3.25GB mmap
- 2.25GB < -Xmx <= 3GB: MAXDATA=0XB0000000@DSA = 3.25GB user space, malloc grows up, mmap grows down
- -Xmx <= 2.25GB: MAXDATA=0XA0000000@DSA = 2.75GB user space, malloc grows up, mmap grows down, shared libraries in 0xD and 0xF
- 0@DSA is not very practical because it only leaves a single segment for native heap (malloc) which is usually insufficient

If you need more native memory (i.e. native OOM but not a leak), and your -Xmx is less than 2.25GB, explicitly setting 0xB@DSA may be useful by increasing available native memory by approximately 400MB to 600MB. This causes the shared/mapped storage to start at 0xF and grow down. The cost is that shared libraries are loaded privately which increases system-wide virtual memory load (and thus potentially physical memory requirements!). If you change X JVMs on one machine to the 0xB@DSA memory model, then the total virtual and real memory usage of that machine may increase by up to (N*(X-1)) MB, where N is the size of the shared libraries’ code and data. Typically for stock WebSphere Application Server, N is about 50MB to 100MB. The change should not significantly affect performance, assuming you have enough additional physical memory.

Another effect of changing to the 0xB@DSA memory model is that segment 0xE is no longer available for mmap/shmat, but instead those allocations grow down in the same way as the Java heap. If your -Xmx is a multiple of 256MB (1 segment), and your process uses mmap/shmat (e.g. client files), then you will have one less segment for native memory. This is because native memory allocations (malloc) cannot share segments with mmap/shmat (Java heap, client files, etc.). To fully maximize this last segment for native memory, you can calculate the maximum amount of memory that is mmapped/shmat’ed at any one time using svmon (find mmapped sources other than the Java heap and clnt files), and then subtract this amount from -Xmx. -Xmx is not required to be a multiple of 256MB, and making room available in the final segment may allow the mmapped/shmatted allocations to be shared with the final segment of the Java heap, leaving the next segment for native memory. This only works if said mmaps/shmats are not made to particular addresses.

Setting 0xB@DSA:

- Admin Console -> Application Servers -> $SERVER -> Java and Process Management -> Process Definition -> Environment Entries
- Click New. Name: LDR_CNTRL
- Value: MAXDATA=0XB0000000@DSA
- Click OK
- Click New. Name: IBM_JVM_LDR_CNTRL_NEW_VALUE
Value: MAXDATA=0XB0000000@DSA
Click OK
Save, synchronize, and restart


### Input/Output (I/O)

#### Disk

Investigate disk performance:

Start `iostat`:

```bash
nohup iostat -DRlt 10 >iostat.txt 2>&1 &
```

Stop `iostat`

```bash
kill $(ps -ef | grep iostat | grep -v grep | awk '{print $2}')
```

Example `iostat` output:

```
System configuration: lcpu=56 drives=2 paths=8 vdisks=0

Disks:                      xfers                                read
write                      queue                                  time
------------------------------------  -----------------------------------
------------------------------------  -----------------------------------
%tm    bps   tps  bread  bwrtn   rps    avg    min    max  avg  serv
avg    serv   serv   serv  outs        time   time   time  wqsz
act                                    serv   serv   serv
outs              serv   serv   serv outs        time   time   time  wqsz
sqsz qfull
hdisk0            0.1  86.4K   2.3   0.0   86.4K   0.0   0.0    0.0    0.0
0 0    0.5 0.3    1.2   0 0   0.0   0.0    0.0   0.0    0.0    0.0
0.0 03:54:59
hdisk1            0.0  86.4K   2.3   0.0   86.4K   0.0   0.0    0.0    0.0
0 0    0.4 0.3    0.8   0 0   0.0   0.0    0.0   0.0    0.0    0.0
0.0 03:54:59
```

Disks:                      xfers                                read
write                      queue                                  time
------------------------------------  -----------------------------------
------------------------------------  -----------------------------------
%tm    bps   tps  bread  bwrtn   rps    avg    min    max  avg  serv
avg    serv   serv   serv  outs        time   time   time  wqsz
act                                    serv   serv   serv
outs              serv   serv   serv outs        time   time   time  wqsz
sqsz qfull
hdisk0            0.9 133.2K  21.3   0.0  133.2K   0.0   0.0    0.0    0.0
0 0    0.3 0.3    0.9   0 0   0.0   0.0    0.0   0.0    0.0    0.0
0.0 03:54:59
```
Consider reducing the inode cache if you observe memory pressure: `ioo -p -o j2_inodeCacheSize=100 -o j2_metadataCacheSize=100`

"The ioo settings for j2 inode cache and meta data cache sizes need to be evaluated on a case by case basis. Determine if the values are too high by comparing the number of client segments in the 'svmon -S' output with the number of unused segments. Also consider the absolute number of client segments. As files are opened, we expect these numbers to go up. Do not adjust anything unless the number of client segments exceeds about 250,000 and the number of unused segments is greater than about 95%. In most cases, reduce them to 100 each." (AIX Expert)

**Networking**

For hostname resolution, by default DNS is tried before `/etc/hosts`, unless DNS is not set up (no `/etc/resolv.conf` file). If you would like to optimize DNS lookup by placing entries into `/etc/hosts`, then consider changing the order of hostname lookup, either through `/etc/irs.conf` or the environment variable NSORDER:


Query network interfaces:

```
# ifconfig -a
en0:
  flags=1e080863,480<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,
  64BIT,CHECKSUM_OFFLOAD(ACTIVE),CHAIN>
  inet 10.20.30.10 netmask 0xffffff00 broadcast 10.20.30.1
  tcp_sendspace 262144 tcp_recvspace 262144 rfc1323 1
```

Query network statistics for a particular interface every 5 seconds:

```
# netstat -I en0 5
input   (en0)      output           input   (Total)    output
  packets  errs  packets  errs colls  packets  errs  packets  errs colls
158479802     0 21545659     0     0 178974399     0 42040363     0     0
  25     0        1     0     0       29     0        5     0     0
  20     0        4     0     0       22     0        6     0     0
```

Tips on network monitoring: http://www-01.ibm.com/support/docview.wss?uid=tss1td105040&aid=1

Query the Maximum Transmission Unit (MTU) of a network adapter:

```
# lsattr -El en1 | grep "^mtu"
mtu       1500       Maximum IP Packet Size for This Device       True
```

Tune various kernel parameters based on the type and MTU size of the adapter:

If dedicated network adapters are set up for inter-LPAR network traffic, recent versions of AIX support super jumbo frames up to 65280 bytes:

```
# chdev -l en1 -a mtu=65280
```

Network throughput can be tested easily with FTP:

```
ftp> put "|dd if=/dev/zero bs=64k count=100000" /dev/null
200 PORT command successful.
150 Opening data connection for /dev/null.
100000+0 records in.
100000+0 records out.
226 Transfer complete.
```

6553600000 bytes sent in **170.2 seconds** (3.761e+04 Kbytes/s)

local: |dd if=/dev/zero bs=64k count=100000 remote: /dev/null

The `netstat` command can be used to query kernel network buffers (http://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.performance/netstat_command_monitor_mbuf.htm):

```
# netstat -m
Kernel malloc statistics:
******* CPU 0 *******
By size           inuse     calls failed   delayed    free   hiwat   freed
64                  778  16552907      0        13     182   10484       0
128                 521   1507449      0        16     183    5242       0...
```

The `failed` and `delayed` columns should be zero.

If using ethernet interfaces, check Packets Dropped: 0, Hypervisor Send Failures, Hypervisor Receive Failures, and Receive Buffer usage for all ethernet interfaces in netstat -v:

```
# netstat -v
...
Hypervisor Send Failures: 0
Hypervisor Receive Failures: 0
Packets Dropped: 0
...
Receive Information
Receive Buffers
   Buffer Type Tiny  Small Medium Large  Huge
Min Buffers  512   512    128  24     24
Max Buffers  2048  2048   256  64     64
Allocated    512   512    128  24     24
Registered   512   512    128  24     24
History      
   Max Allocated    512   1138  128  24     24
   Lowest Registered 506   502    128  24     24
```

If "Max Allocated" for a column is greater than "Min Buffers" for that column, this may have caused reduced performance. Increase the buffer minimum using, for example:

```
# chdev -P -l ${INTERFACE} -a min_buf_small=2048
```
If "Max Allocated" for a column is equal to "Max Buffers" for that column, this may have caused dropped packets. Increase the buffer maximum using, for example:

```bash
# chdev -P -l ${INTERFACE} -a max_buf_small=2048
```

NOTE: It is necessary to bring down the network interface(s) and network device(s) changed by the above commands and then restart those devices and interfaces. Some customers prefer to simply reboot the LPAR after running the command(s).


VIO VEA buffers may need to be increased when you run `entstat -d ${INTERFACE}`:

```
ETHERNET STATISTICS (ent0) :
Hypervisor Send Failures: 58749
Receiver Failures: 58749
Send Errors: 0
Hypervisor Receive Failures: 4713
```

<table>
<thead>
<tr>
<th>Buffer Type</th>
<th>Tiny</th>
<th>Small</th>
<th>Medium</th>
<th>Large</th>
<th>Huge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min Buffers</td>
<td>512</td>
<td>512</td>
<td>128</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Max Buffers</td>
<td>2048</td>
<td>2048</td>
<td>256</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Allocated</td>
<td>512</td>
<td>512</td>
<td>128</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>Max Allocated</td>
<td>512</td>
<td>592</td>
<td>128</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

If the "Failures" or "Errors" lines are non-zero, then increase the VEA buffers and reboot:

```bash
chdev -l ent0 -a max_buf_small=4096 -a min_buf_small=4096 -P
```

A symptom of this might be when a non-blocking `write` appears to block with low CPU, whereas it would normally block in `poll`.

**tcpdump**


Normally, tcpdump is run as root. For example, capture all traffic in files of size 100MB and up to 10 historical files (-C usually requires -Z):

```bash
$ su
# nohup tcpdump -n -i ${INTERFACE} -s 0 -C 100 -Z root -w capture hostname`_`date +"%Y%m%d_%H%M"`.dmp &
# sleep 1 && cat nohup.out
```

To stop the capture:

```bash
$ su
# ps -elf | grep tcpdump | grep -v grep | awk '{print $4}' | xargs kill -INT
```

Use Wireshark to analyze (covered in the [Major Tools chapter](http://www-01.ibm.com/support/docview.wss?uid=swg21175744)).

**iptrace**

Start capturing all traffic:
   # iptrace -a -b trace.out

To limit the size of the file, use the -L ${BYTES} option which will roll to a single historical file. For example, the following limits to 512MB:
   # iptrace -a -b trace.out -L 536870912

To limit the bytes captured per packet, use the -S ${BYTES} option. For example, the following limits each packet to 96 bytes:
   # iptrace -a -b trace.out -S 96

Stop capturing traffic:
   # ps -elf | grep iptrace | grep -v grep | awk '{print $4}' | xargs kill -15

Use Wireshark to analyze (covered in the Operating Systems chapter).

Filter to only capture traffic coming into or going out of port 80:
   # iptrace -a -b -p 80 trace.out

Ping a remote host. In general, and particularly for LANs, ping times should be less than a few hundred milliseconds with little standard deviation.

   $ ping -n 10.20.30.1
   PING 10.20.30.1: (10.20.30.1): 56 data bytes
   64 bytes from 10.20.30.1: icmp_seq=0 ttl=241 time=63 ms
   64 bytes from 10.20.30.1: icmp_seq=1 ttl=241 time=63 ms
   64 bytes from 10.20.30.1: icmp_seq=2 ttl=241 time=63 ms
   64 bytes from 10.20.30.1: icmp_seq=3 ttl=241 time=63 ms

**Kernel Parameters**

The "no" command is used to query or set network related kernel parameters. To display current values:
   # no -a

To update a value, use -o, for example:
   # no -o tcp_timewait=1

However, this only lasts until reboot. To make this change permanent, use the -r flag (this will update the /etc/tunables/nextboot file):
   # no -r -o tcp_timewait=1

You may query the default value of a parameter using no -L:
   # no -L tcp_nodelayack

```
NAME                      CUR    DEF    BOOT   MIN    MAX    UNIT
---                      ----    ---    ----    -----    ----    ----
```
Update the TIME_WAIT timeout to 15 seconds by running # no -r -o tcp_timewait=1

Update the AIX ARP Table Bucket Size by running # no -r -o arptab_size=10

Update the interval between TCP keepalive packets by running # no -r -o tcp_keepintvl=10

Update the initial timeout for TCP keepalive by running # no -o tcp_keepinit=40

Recent versions of AIX include a TCP Traffic Regulation (TR) feature which is designed to protect against network attacks. By default it is off, but security hardening commands such as aixpert may enable it indirectly. If you are experiencing mysterious connection resets at high load (a message is not logged when this function is exercised), this may be working as designed and you can tune or disable this function using the tcptr and no commands:

Consider tuning interrupt coalescing/moderation:

Ensure that all network interfaces are either explicitly set to maximum speed or they are auto-negotiating maximum speed. Also ensure that the speed is full duplex:

First, get a list of the interfaces: ifconfig -a

Example:

$ ifconfig -a
en0:
  flags=1e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD(ACTIVE),LARGESEND,CHAIN-
inet 153.10.11.11 netmask 0xffffff00 broadcast 153.10.11.255
tcp_sendspace 262144 tcp_recvspace 262144 rfc1323 1
en1:
  flags=1e080863,c0<UP,BROADCAST,NOTRAILERS,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD(ACTIVE),LARGESEND,CHAIN-
inet 143.10.11.11 netmask 0xffffff00 broadcast 143.10.11.255
tcp_sendspace 131072 tcp_recvspace 65536 rfc1323 0
lo0:
  flags=e08084b,c0<UP,BROADCAST,LOOPBACK,RUNNING,SIMPLEX,MULTICAST,GROUPRT,64BIT,CHECKSUM_OFFLOAD(ACTIVE),LARGESEND,CHAIN-
inet 127.0.0.1 netmask 0xff000000 broadcast 127.255.255.255
tcp_sendspace 102400 tcp_recvspace 102400 rfc1323 0
Next, for each interface that will be used, query the running speed: `entstat -d en0 | grep "Media Speed"

Example:

$ entstat -d en0 | grep "Media Speed"
Media Speed Selected: Autonegotiate
Media Speed Running: 10000 Mbps / 10 Gbps, Full Duplex

To disable Nagle's algorithm, you can either:

1. Disable it using # no -r -o tcp_nagle_limit=0:
2. Disable it at an interface level using tcp_nodelay=1:

**Other Kernel and Process Settings**

Update the maximum open files ulimit to 10,000 by adding the following lines to `/etc/security/limits` for the relevant user accounts (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tuneaix.html):

```
nofiles = 10000
nofiles_hard = 10000
```


As root user, run the following commands to reserve 4 GB of large page:

```
# vmo -r -o lgpg_regions=256 -o lgpg_size=16777216
# bosboot -ad /dev/ipldevice
# reboot -q
```

After reboot, run the following command to enable large page support on the AIX operating system:

```
# vmo -p -o v_pinshm=1
```

As root user, add the following capabilities for the relevant users:

```
# chuser capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE $USER
```

Processor sets/pinning

The AIX scheduler generally does a good job coordinating CPU usage amongst threads and processes, however manually assigning processes to CPUs can provide more stable, predictable behavior. Binding processes to particular CPUs is especially important on systems with multiple processing modules and non-uniform memory access (see the next section on memory affinity), and also depending on how various levels of cache are shared between processors. It is best to understand the system topology and partition resources accordingly, especially when multiple CPU intensive processes must run on the machine. The easiest way to do this is using the execrset command to specify a list of CPUs to bind a command (and its children) to:

```
execrset -c <CPUs> -e <command>
```

(execrset requires the CAP_NUMA_ATTACH property when running as non-root.)


Note that on SMT-enabled machines the list of CPUs will represent logical CPUs. For example, if the machine was booted in SMT4 mode, CPUs 0-3 represent the 4 hardware threads that the physical CPU 0 can support. The current SMT mode and logical to physical mappings can be queried using the smtctl command.

It is important to note that currently the J9 JVM configures itself based on the number of online processors in the system, not the number of processors it is bound to (which can technically change on the fly). Therefore, if you bind the JVM to a subset of CPUs you should adjust certain thread-related options, such as -Xgcthreads, which by default is set to the number of online processors.

Memory Affinity

Memory affinity can be an important consideration when dealing with large systems composed of multiple processors and memory modules. POWER-based SMP systems typically contain multiple processor modules, each module housing one or more processors. Each processing module can have a system memory chip module (MCM) attached to it, and while any processors can access all memory modules on the system, each processor has faster access to its local memory module. AIX memory affinity support allows the OS to allocate memory along module boundaries and is enabled by default in AIX 5.2. To enable/disable it explicitly, use vmo -o memory_affinity=1/0.

If memory affinity is enabled the default memory allocation policy is a round-robin scheme that rotates allocation amongst MCMs. Using the environment variable MEMORY_AFFINITY=MCM will change the policy to allocate memory from the local MCM whenever possible. This is especially important if a process has been bound to a subset of processors, using execrset for example; setting MEMORY_AFFINITY=MCM will reduce the amount of memory allocated on non-local MCMs and improve performance.
Disabling Hardware Prefetching

The dscrctl command sets the hardware prefetching policy for the system. Hardware prefetching is enabled by default and is most effective when memory access patterns are easily predictable. The hardware prefetcher can be configured with various schemes, however most transaction oriented Java workloads may not benefit from hardware prefetching so you may see improved performance by disabling it using dscrctl -n -s 1. Starting with Java 6.0.1 sr4 and Java 7 sr3, the JVM provides the -XXsetHWPrefetch command-line switch to set the hardware prefetch policy for its process only. Use -XXsetHWPrefetch:none to disable prefetching and -XXsetHWPrefetch=N to enable a specific prefetch policy, where N is a value recognized by dscrctl. Starting with Java 6.0.1 sr5 and Java 7 sr4, the JVM will disable hardware prefetching by default, use -XXsetHWPrefetch:os-default to revert to the previous behavior and allow the JVM process to use the policy currently set with dscrctl.

Also consider the option -XnotlhPrefetch

Native Memory Allocation (malloc) Algorithms

At one customer, we improved throughput by 50% simply by restarting with the AIX environment variable MALLOCOPTIONS=multiheap (http://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.genprogc/malloc_multiheap.htm). This only applies to situations where there is heavy, concurrent malloc usage, and in many cases of WAS/Java, this is not the case.

The multiheap option does have costs, particularly increased virtual and physical memory usage. The primary reason is that each heap's free tree is independent, so fragmentation is more likely. There is also some additional metadata overhead.

malloc is often a bottleneck for application performance, especially under AIX... By default, the [AIX] malloc subsystem uses a single heap, which causes lock contention for internal locks that are used by malloc in case of multi-threaded applications. By enabling [the multiheap] option, you can configure the number of parallel heaps to be used by allocators. You can set the multiheap by exporting MALLOCOPTIONS=multipheap[:n], where n can vary between 1-32 and 32 is the default if n is not specified. Use this option for multi-threaded applications, as it can improve performance.


Increasing the number of malloc heaps does not significantly increase the virtual memory usage directly (there are some slight increases because each heap has some bookkeeping that it has to do). However, each heap's free tree is independent of others, but the heap areas all share the same data segment, so native memory fragmentation becomes more likely, and thus indirectly virtual and physical memory usage may increase. It is impossible to predict by how much because it depends on the rate of allocations and frees, sizes of allocations, number of threads, etc. It is best to take the known physical and virtual memory usage of a process before the change (rss, vsz) at peak workload, so let's call this X GB (for example, 9 GB). Then apply the change and run the process to peak workload and monitor. The additional usage will normally be no more than 5% of X (in the above example, ~500MB). As long as there is that much additional physical memory available, then things should be okay. It is advised to continue to monitor rss/vsz after the change, especially over time (fragmentation has a tendency to build up).
How do you know if this is affecting you? It's not easy:

A concentration of execution time in the pthreads library... or in kernel locking... routines... is associated with a locking issue. This locking might ultimately arise at the system level (as seen with malloc locking issues on AIX), or at the application level in Java code (associated with synchronized blocks or methods in Java code). The source of locking issues is not always immediately apparent from a profile. For example, with AIX malloc locking issues, the time that is spent in the malloc and free routines might be quite low, with almost all of the impact appearing in kernel locking routines.

Nevertheless, here is an example tprof that shows this problem using `tprof -ujeskzl -A -I -X -E -r report -x sleep 60`:

<table>
<thead>
<tr>
<th>Process</th>
<th>FREQ</th>
<th>Total Kernel</th>
<th>User</th>
<th>Shared</th>
<th>Other</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/java5/jre/bin/java</td>
<td>174</td>
<td>22557</td>
<td>11850</td>
<td>0</td>
<td>7473</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared Object</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libc.a[shr_64.o]</td>
<td>3037</td>
<td>9.93</td>
<td>900000000000d00</td>
<td>9000000007fe200</td>
<td>331774</td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libpthread.a[shr_xpg5_64.o]</td>
<td>1894</td>
<td>6.19</td>
<td>9000000000007fe200</td>
<td>319a8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total Ticks For All Processes (KERNEL) = 15045

<table>
<thead>
<tr>
<th>Process</th>
<th>FREQ</th>
<th>Total Kernel</th>
<th>User</th>
<th>Shared</th>
<th>Other</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>/usr/lib/libc.a[shr_64.o]</td>
<td>3037</td>
<td>9.93</td>
<td>900000000000d00</td>
<td>9000000007fe200</td>
<td>331774</td>
<td></td>
</tr>
<tr>
<td>/usr/lib/libpthread.a[shr_xpg5_64.o]</td>
<td>1894</td>
<td>6.19</td>
<td>9000000000007fe200</td>
<td>319a8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The key things to notice are:

1. In the first "Process" section, the "Kernel" time is high (about half of "Total"). This will also show up in topas/vmstat/ps as high "system" CPU time.
2. In the "Shared Object" list, libc and libpthread are high.
3. In the "KERNEL" section, ._check_lock is high.
4. In the "libc.a" section, .malloc_y and .free_y are high.
5. In the "libpthread.a" section, .global_unlock_ppc_mp and other similarly named functions are
If you see a high percentage in the KERNEL section in "unlock_enable_mem," this is usually caused by calls to sync 1/sync L/lwsync (http://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.alangref/idalangref_sync_dcs_instrs.htm, http://www.ibm.com/developerworks/systems/articles/powerpc.html). It has been observed in some cases that this is related to the default, single threaded malloc heap.

AIX also offers other allocators and allocator options that may be useful:

- **Pool malloc**: The pool front end to the malloc subsystem optimizes the allocation of memory blocks of 512 bytes or less. It is common for applications to allocate many small blocks, and pools are particularly space- and time-efficient for that allocation pattern. Thread-specific pools are used for multi-threaded applications. The pool malloc is a good choice for both single-threaded and multi-threaded applications.

- **Using the pool front end and multiheap malloc in combination is a good alternative for multi-threaded applications. Small memory block allocations, typically the most common, are handled with high efficiency by the pool front end. Larger allocations are handled with good scalability by the multiheap malloc. A simple example of specifying the pool and multiheap combination is by using the environment variable setting:**

    MALLOCOPTIONS=pool,multiheap

- **Buckets**: This suboption is similar to the built-in bucket allocator of the Watson allocator. However, with this option, you can have fine-grained control over the number of buckets, number of blocks per bucket, and the size of each bucket. This option also provides a way to view the usage statistics of each bucket, which can be used to refine the bucket settings. In case the application has many requests of the same size, then the bucket allocator can be configured to preallocate the required size by correctly specifying the bucket options. The block size can go beyond 512 bytes, compared to the Watson allocator or malloc pool options.

1. For a 32-bit single-threaded application, use the default allocator.
2. For a 64-bit application, use the Watson allocator.
3. Multi-threaded applications use the multiheap option. Set the number of heaps proportional to the number of threads in the application.
4. For single-threaded or multi-threaded applications that make frequent allocation and deallocation of memory blocks smaller than 513, use the malloc pool option.
5. For a memory usage pattern of the application that shows high usage of memory blocks of the same size (or sizes that can fall to common block size in bucket option) and sizes greater than 512 bytes, use the configure malloc bucket option.
6. For older applications that require high performance and do not have memory fragmentation issues, use malloc 3.1.
7. Ideally, the Watson allocator, along with the multiheap and malloc pool options, is good for most multi-threaded applications; the pool front end is fast and is scalable for small allocations, while multiheap ensures scalability for larger and less frequent allocations.
8. If you notice high memory usage in the application process even after you run free(), the
disclaim option can help.

Large page support available for applications that can take advantage of it.


### Useful Commands


### z/OS


### z/OS Recipe

1. **CPU core(s)** should not be consistently saturated.
2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.
9. Use the Workload Activity Report to review performance.

Also review the general topics in the Operating Systems chapter.

### General

The z/OS graphical user interface is normally accessed through a 3270 session. A commonly used Linux client program is x3270:
If configured, z/OS is often also accessed through Telnet, SSH, or FTP.

On some 3270 client keyboard layouts, the right "Ctrl" key is used as the "Enter" key.

z/OS uses the EBCDIC character set by default instead of ASCII/UTF; however, some files produced by Java are written in ASCII or UTF. These can be converted using the iconv USS command or downloaded through FTP in BINARY mode to an ASCII/UTF based computer.

**Interactive System Productivity Facility (ISPF)**

After logging in through a 3270 session, it is common to access most programs by typing ISPF. Typically, if available, F7 is page up, F8 is page down, F10 is page left, and F11 is page right. Typing "m" followed by F7 or F8 pages down to the top or bottom, respectively.

If available, type "F STRING" to find the first occurrence of STRING, and F5 for the next occurrences.

Normally, F3 returns to the parent screen (i.e. exits the current screen).

Command (normally type 6 in ISPF) allows things such as TSO commands or going into Unix System Services (USS).

Utilities - Data Set List (normally type 3.4 in ISPF) allows browsing data sets.

**System Display and Search Facility (SDSF)**

SDSF (normally type S in ISPF) provides a system overview.

**SDSF.LOG**

LOG shows the system log and it is the most common place to execute system commands. Enter a system command by pressing /, press enter, and then type the command and press enter. Then use F8 or press enter to refresh the screen to see the command’s output.

Display system activity summary: /D A

```
IEE114I 16.18.32 2011.250 ACTIVITY 733
JOBS M/S TS USERS SYSAS INITS ACTIVE/MAX VTAM OAS
00008 00034 00001 00035 00034 00001/00300 00019
```

Display users on the system: /D TS,L

```
IEE114I 16.51.50 2011.251 ACTIVITY 298
JOBS M/S TS USERS SYSAS INITS ACTIVE/MAX VTAM OAS
00008 00039 00002 00039 00034 00002/00300 00029
DOUGMAC OWT WITADM1 IN
```

Check global resource contention with /D GRS,C

**SDSF.DA**

SDSF.DA shows active address spaces
"CPU/L/Z A/B/C" shows current CPU use, where A=total, B=LPAR usage, and C=zAAP usage.
Type "PRE *" to show all address spaces.
Type "SORT X" to sort, e.g. 'SORT CPU%'.
Page right to see useful information such as MEMLIMIT, RPTCLASS, WORKLOAD, and SRVCLASS.
In the NP column, type "S" next to an address space to get all of its output, or type ? to get a member list and then type S for a particular member (e.g. SYSOUT, SYSPRINT).
When viewing joblog members of an address space (? in SDSF.DA), type XDC next to a member to transfer it to a data set.
SDSF.ST is similar to DA and includes completed jobs.

**z/OS Version**

Display z/OS version with /D IPLINFO

Search for the "RELEASE" line:

```
IEE254I 13.06.07 IPLINFO DISPLAY 033
SYSTEM IPLED AT 09.38.57 ON 05/15/2018
RELEASE z/OS 02.02.00 LICENSE = z/OS
USED LOADRE IN SYS1.IPLPARM ON 00340
ARCHLVL = 2 MTLSHARE = N
IEASYM LIST = (05,RE,L)
IEASYS LIST = (LF,KB) (0P)
IODF DEVICE: ORIGINAL(00340) CURRENT(00340)
IPL DEVICE: ORIGINAL(00980) CURRENT(00980) VOLUME(PDR22 )
```

**Central Processing Unit (CPU)**

Display processors: /D M=CPU

```
D M=CPU
IEE174I 15.45.46 DISPLAY M 700
PROCESSOR STATUS
ID CPU SERIAL
00 + 0C7B352817
01 + 0C7B352817
02 + 0C7B352817
03 + 0C7B352817
04 +A 0C7B352817
05 +A 0C7B352817
06 +A 0C7B352817
07 +A 0C7B352817
+ ONLINE - OFFLINE . DOES NOT EXIST W WLM-MANAGED N NOT AVAILABLE A APPLICATION ASSIST PROCESSOR (zAAP)
```

This shows four general purpose processors and four zAAP processors.
Display threads in an address space and the accumulated CPU by thread: /D OMVS,PID=XXX (http://www-01.ibm.com/support/docview.wss?uid=tss1wp101474&aid=1). You can search for PID in the joblogs of the address space. This includes a CT_SECS field which shows the total CPU seconds consumed by the address space. Note that the sum of all the ACC_TIME in the report will not equal CT_SECS or the address CPU as reported by RMF or SDSF because some threads may have terminated. The ACC_TIME and CT_SECS fields wrap after 11.5 days and will contain ***** therefore the D OMVS,PID= display is less useful when the address space has been running for longer than that.

```
-BPX0040I 11.09.56 DISPLAY OMVS 545
OMVS 000F ACTIVE OMVS=(00,FS,0A)
USER JOBNAME ASID PID PPID STATE START CT_SECS
WSASRU WSODRA S 0190 67502479 1 HR---- 23.01.38 13897.128 1
LATCHWAITPID= 0 CMD=BB0SR
THREAD_ID        TCB@    PRI_JOB USERNAME ACC_TIME SC STATE
2621F4D000000008 009C7938                       12.040 PTX JY V
```

All the threads/TCBs are listed and uniquely identified by their thread ID under the THREAD_ID column. The accumulated CPU time for each thread is under the ACC_TIME column. The thread ID in OMVS,PID is the first 8 hexadecimal characters in the THREAD_ID and can be found in a javacore.txt file. In the example above, the Java thread ID is 2621F4D0.

The threads with eye-catcher “WLM” are those from the ORB thread pool which are the threads that run the application enclave workload. Be careful when attempting to reconcile these CPU times with CPU accounting from RMF and SMF. This display shows all the threads in the address space, but remember that threads that are WLM managed (e.g. the Async Worker threads and the ORB threads) have their CPU time recorded in RMF/SMF under the enclave which is reported in the RMF report class that is associated with the related WLM classification rule for the "CB" workload type. The other threads will have their CPU time charged to the address space itself as it is classified in WLM under the "STC" workload type.

WebSphere trace entries also contain the TCB address of the thread generating those entries. For example:

```
THREAD_ID TCB@ PRI_JOB USERNAME ACC_TIME SC STATE
2707058000000078 009BDB58 178.389 STE JY V
Trace: 2009/03/19 08:28:35.069 01 t=9BDB58 c=UNK key=P8 (0000000A)
```

The SDSF.PS display provides an easy way to issue this command for one or more address spaces. Type "d" next to an address space to get this same output. Type "ULOG" to see the full output or view in SDSF.LOG.

Similar information can be found with: ps -p <pid_number> -m -o xtid,xtcbaddr,tagdata,state=STATE -o atime=CPUTIME -o syscall

```
TID TCBADDR STATE CPUTIME SYSC
- - HR 14:12 -
1e4e300000000000 8d0e00 YU 0:20
1e4e400000000001 8d07a8 YJV 0:00
1e4e500000000002 8d0588 YNJV 0:00
1e4e600000000003 8d0368 YJV 1:35
1e4e700000000004 8d0148 YJV 0:25
```
31-bit vs 64-bit

z/OS does not have a 32-bit architecture, but instead only has a 31-bit architecture:

![Diagram showing address space comparison between 31-bit and 64-bit modes]

In 31-bit mode, address space is 2GB.
- Typical MVS system, private region size is 1.4GB.
- Subtract infrastructure and max usable JVM heap is between 768 and 900MB

In 64-bit mode, address space is 16EB.
- Allows for larger JVM heap, if you want one
- Consider real and auxiliary

Diagram not to scale!
- 64bit address space 8,589,934,592 x larger
- If 31bit picture were 2 inches / 5 cm high
  ...64bit would extend past the moon
  ...even after subtracting system overhead

(https://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/44f15b38101bcc6e8625737c006987c5/$FILE/WP101121-%20The%-2064bits%-20Effect%-20%-20Five%-20Different%-20Ways%-20to%-20Look%-E2%-80%-A6.pdf)

zAAP Processors

Review zAAP processors:

1. Type /D IPLINFO and search for LOADY2.
2. Go to the data set list and type the name from LOADY2 in Dsname level and press enter (e.g. SYS4.IPLPARM).
3. Type ‘b’ to browse the data set members and search for PARMLIB.
4. Go to the data set list and type the name (e.g. USER.PARMLIB) and find the IEAOPT member.

Inside SYS1.PARMLIB(IEAOPTx), the following options will affect how the zAAP engines process work.

1. IFACrossOver = YES / NO
   - YES - work can run on both zAAP and general purpose CPs
   - NO – work will run only on zAAPs unless there are no zAAPs
2. IFAHonorPriority = YES / NO
   - YES – WLM manages the priority of zAAP eligible work for CPs
   - NO – zAAP eligible work can run on CPs but at a priority lower than any non-zAAP work
Physical Memory (RAM)

Use /D M=STOR to display available memory. The ONLINE sections show available memory. For example, this shows 64GB:

```
D M=STOR
IEE174I 16.00.48 DISPLAY M 238
REAL STORAGE STATUS
ONLINE-NOT RECONFIGURABLE
  0M-64000M
ONLINE-RECONFIGURABLE
  NONE
PENDING OFFLINE
  NONE
0M IN OFFLINE STORAGE ELEMENT(S)
0M UNASSIGNED STORAGE
STORAGE INCREMENT SIZE IS 256M
```

Use /D ASM to display paging spaces. The FULL columns for LOCAL entries should never be greater than 0%. For example:

```
D ASM
IEE200I 15.30.16 DISPLAY ASM 205
TYPE   FULL STAT  DEV DATASET NAME
PLPA    79%   OK 0414 SYS1.S12.PLPA
COMMON  0%   OK 0414 SYS1.S12.COMMON
LOCAL   0%   OK 0414 SYS1.S12.LOCAL1
LOCAL   0%   OK 0445 SYS1.S12.PAGE01
```

Display total virtual storage: /D VIRTSTOR,HVSHARE

```
IAR019I 17.08.47 DISPLAY VIRTSTOR 313
SOURCE = DEFAULT
TOTAL SHARED = 522240G
SHARED RANGE = 2048G-524288G
SHARED ALLOCATED = 262244M
```

Some systems display free memory with /F AXR,IAXDMEM:


```
IAR049I DISPLAY MEMORY V1.0 233
PAGEABLE 1M STATISTICS
  66.7GB : TOTAL SIZE
  50.8GB : AVAILABLE FOR PAGEABLE 1M PAGE
  2404.0MB : IN-USE FOR PAGEABLE 1M PAGES
  5238.0MB : MAX IN-USE FOR PAGEABLE 1M PAGES
  0.0MB : FIXED PAGEABLE 1M FRAMES
LFAREA 1M STATISTICS - SOURCE = DEFAULT
  0.0MB : TOTAL SIZE
  0.0MB : AVAILABLE FOR FIXED 1M PAGES
  0.0MB : IN-USE FOR FIXED 1M PAGES
  0.0MB : MAX IN-USE FOR FIXED 1M PAGES
LFAREA 2G STATISTICS - SOURCE = DEFAULT
  0.0MB : TOTAL SIZE = 0
```
Job Entry Subsystem (JES)

Use /$DSPOOL to list spool utilization. For example:

$HASP646  41.0450 PERCENT SPOOL UTILIZATION

Workload Management (WLM)

WLM only makes noticeable decisions about resources when resources are low.

WLM performs better with less service classes.

- Service Classes – goals for a particular type of work – you can have as many of these as you want but from a performance perspective the fewer service classes the better
- Classification Rules – classification rules tie an address space or group of address spaces to a goal or service class
- Report Classes – report classes have nothing to do with classification of work but they do allow you to show reports from a particular perspective for problem and performance diagnosis

Display WLM configuration: /D WLM

IWM025I  14.31.46  WLM DISPLAY 214
ACTIVE WORKLOAD MANAGEMENT SERVICE POLICY NAME: CBPTILE
ACTIVATED: 2011/06/13  AT: 16:15:27  BY: WITADM1   FROM: S12
DESCRIPTION: CB trans w/short percentile goal
RELATED SERVICE DEFINITION NAME: CBPTILE
INSTALLED: 2011/06/13  AT: 16:15:08  BY: WITADM1   FROM: S12

The related service definition name is the currently configured WLM definition.

Classify location service daemons and controllers as SYSSTC or high velocity.

Set achievable percentage response time goals: For example, a goal that 80% of the work will complete in .25 seconds is a typical goal. Velocity goals for application work are not meaningful and should be avoided.

Make your goals multi-period: This strategy might be useful if you have distinctly short and long running transactions in the same service class. On the other hand, it is usually better to filter this work into a different service class if you can. Being in a different service class will place the work in a different servant which allows WLM much more latitude in managing the goals.

Define unique WLM report classes for servant regions and for applications running in your
application environment. Defining unique WLM report classes enables the resource measurement facility (RMF) to report performance information with more granularity.

Periodically review the results reported in the RMF Postprocessor workload activity report: Transactions per second (not always the same as client tran rate), Average response times (and distribution of response times), CPU time used, Percent response time associated with various delays

Watch out for work that defaults to SYSOTHER.


Delay monitoring:

Example:

You can print the entire WLM definition from the main screen:

Within the Subsystem types section you will find the classification rules that tie the address spaces to the service classes and report classes. You can also find this by paging right in SDSF.DA.

So what is the Response Time Ratio and what does it tell us? WLM calculates the Response Time Ratio by dividing the actual response time (enclave create to enclave delete) by the GOAL for this service class and multiplying by 100. It is, basically, a percentage of the goal. Note that WLM caps the value at 1000 so if the goal is badly missed you might see some big numbers but they will never exceed 1000. (http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/0c808594b1db5c6286257bb1006118ab/$FILE/ATTHSSAD.pdf/WP102311_SMF_Analysis.pdf)

**System Management Facilities (SMF)**

SMF captures operating system statistics to data sets.

Display what SMF is currently recording: /D SMF,O
The MEMBER is the PARMLIB member holding the configuration. The SYS line shows which SMF types are being monitored. INTVAL is the recording interval (in minutes). The DSNAME members are the working data sets for the SMF data.

Modify the recording interval dynamically: `/RMF,MODIFY ZZ,SYNC(RMF,0),INTERVAL(15M)`

Display SMF data set usage: `/D SMF`

<table>
<thead>
<tr>
<th>NAME</th>
<th>VOLSER</th>
<th>SIZE(BLKS)</th>
<th>%FULL</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-SYS1.S12.MANC</td>
<td>SMF001</td>
<td>180000</td>
<td>79</td>
<td>ACTIVE</td>
</tr>
<tr>
<td>S-SYS1.S12.MAND</td>
<td>SMF001</td>
<td>180000</td>
<td>0</td>
<td>ALTERNATE</td>
</tr>
</tbody>
</table>

When the active volume fills up, SMF switches to the alternative. This can be done manually with `/I SMF`

**Example JCL to Dump SMF**

```jcl```
//SMFD3 JOB MSGCLASS=H,MSGLEVEL=(1,1),REGION=128M,TIME=5,
// NOTIFY=&SYSUID
// SET SMFIN=S25J.ZTESTB.S12.SMF.G0213V00
// * OUTPUT DATASET NAME
// SET DUMPOUT=ZPER.WM0.SMFS12.D213
// *
//S0      EXEC PGM=IFASMFDP,REGION=128M
//SYSPRINT DD  SYSOUT=* 
//DUMPIN1      DD DISP=SHR,DSN=&SMFIN
//DUMPOUT      DD DISP=(,CATLG,DELETE),UNIT=SYSDA,
//             SPACE=(CYL,(400,100),RLSE),
//             DSN=&DUMPOUT,
//             LRECL=32760,BLKSIZE=23467,RECFM=VBS
//SYSIN        DD *
//INDD(DUMPIN1,OPTIONS(DUMP))
//OUTDD(DUMPOUT,TYPE(0:255))
/*```

**Example JCL to Dump Live SMF Data Sets into a Permanent One**

```jcl```
//SMFD3 JOB MSGCLASS=H,MSGLEVEL=(1,1),REGION=128M,TIME=5,
// NOTIFY=&SYSUID
// SET SMFIN=S25J.ZTESTG.S34.SMF.G1017V00
// * OUTPUT DATASET NAME
// SET DUMPOUT=ZPER.S34.MEVERET.D092211.A
```
The output from the JCL contains the types of records and number of records in the raw data:

IFA020I DUMP OUT -- ZPER.S34.MEVERET.D092211.A
IFA020I DUMP IN1 -- S25J.ZTESTG.S34.SMF.G1017V00

SUMMARY ACTIVITY REPORT
START DATE-TIME 09/22/2011-09:33:34 END DATE-TIME
RECORD RECORDS PERCENT AVG. RECORD MIN. RECORD MAX. RECORD
TYPE READ OF TOTAL LENGTH LENGTH
2 1 .00 % 18.00
3 1 .00 % 18.00
...
TOTAL 42,572 100 % 1,233.27 18

NUMBER OF RECORDS IN ERROR 0

Example JCL to Dump SMF

//SMFR1 JOB MSGLEVEL=(1,1),MSGCLASS=H
//WKLD@PGP EXEC PGM=EBRSMFPP,REGION=0K
//MFPINPUT DD DSN=ZPER.WM0.SMFS12.D203,DISP=SHR
//PPXSRSPTS DD SYSOUT=*,DCB=(RECFM=FBA,LRECL=133)
//MFPMSGDS DD SYSOUT=* //SYSOUT DD SYSOUT=* //SYSSIN DD *
ETOD(0000,2400)
PTOD(0000,2400)
RTOD(0000,2400)
STOD(0000,2400)
SYSRPTS(WLMGL(RCLASS(W*)))
SYSOUT(H)
/*
SYSRPTS(WLMGL(SCPER,RCLASS(WT7*)))
*/

See also
Example JCL to Clear SMF

```
//SMFCLEAR JOB MSGLEVEL=(1,1)
//STEP1 EXEC PGM=IFASMFDP
//DUMPIN DD DSN=SYS1.S12.MANC,DISP=SHR
//*
//* SYS1.S34.MAN1
//* SYS1.S34.MAN2
//*
//*DUMPIN DD DSN=SYS1.S12.MANC,DISP=SHR
//DUMPOUT DD DUMMY
//SYSPRINT DD SYSOUT=*  
//SYSIN DD *
   INDD(DUMPIN,OPTIONS(CLEAR))
   OUTDD(DUMPOUT,TYPE(000:255))
```

Resource Measurement Facility (RMF)

- Display if RMF ZZ monitor is running: /F RMF,D ZZ
- Start RMF ZZ monitor: /F RMF,S ZZ
- Start RMFGAT: /F RMF,S III

Monitoring RMF in live mode can be very useful (navigate through ISPF). F10 and F11 page backwards and forwards through time.
Workload Activity Report

The JCL to produce this was covered above.

Example snippet output:

<table>
<thead>
<tr>
<th>REPORT BY: POLICY=STANDARD</th>
<th>REPORT CLASS=P00CB</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIPTION =</td>
<td></td>
</tr>
</tbody>
</table>

-TRANSACTIONS- TIME HH.MM.SS.TT -DASD I/O--- ---SERVICE- SERVICE TIME ---APPL %-- -PROMOTED--STORAGE-- ---

**AVG** 0.01 ACTUAL 7 SSCRHT 0.0 10C 0 CPU 2.171 CP 0.05 BLK 0.000 AVG 0.00
**MPL** 0.01 EXECUTION 6 RESBP 0.0 CPU 90465 SRB 0.000 AAPCP 0.00 ENQ 0.000 TOTAL 0.00
**ENDED** 622 QUEUED 0 CONN 0.0 MREQ 0 RCT 0.000 IIPCP 0.00 CMN 0.000 SHARED 0.00
**END/s** 2.07 R/S AFIN 0 DISC 0.0 SRB 0 IIT 0.000 LCK 0.000
**#SNAPS** 0 INELIGIBLE 0 Q-PEND 0.0 TOT 90465 HST 0.000 AAP 0.67
**EXCTD** 0 CONVERSION 0 IOSQ 0.0 /SEC 302 AAP 2.015 IIP N/A SINGLE 0.0
**AVG ENC** 0.01 STD DEV 2 IIP N/A SLOCK 0.0
**REM ENC** 0.00 ABRSPTN 22K IIP N/A SHARED 0.0
**MG ENC** 0.00 TRX_SERV22K IIP N/A HSP 0.0

Important values:

1. **CPU** – this is the total amount of processor time (excluding SRB time), used during this interval. It includes time spent on general purpose CPUs, zAAPs and zIIPs.
2. **SRB** – this is the amount of processor time consumed by SRBs during the interval. An SRB is a special unit of work used primarily by the operating system to schedule functions that need to run quickly and with high priority.
3. **AAP** – this is the amount of time work was running on zAAP processors during the interval. The IIP field is exactly the same as AAP except it reports time spent on zIIP processors. On our system there were no zIIP processors defined so it will be ignored.
4. **Ended** – this is the total number of WebSphere requests that completed during the interval.
5. **CP** – this value represents the amount of time spent on general purpose processor. It includes the CP time and the zAAP time that is reported under the “SERVICE TIME” heading, fields CPU and SRB.
   
   The length of this interval is 5 minutes or 300 seconds so using the CP field value under the “APPL %” heading the amount of CP time is:
   
   \[(CP \times \text{interval length}) / 100 \text{ or } (0.20 \times 300) / 100 = 0.600 \text{ (rounding error)}\]

6. **AAPCP** – this value is the amount of zAAP time that ran on a CP which could have run on a zAAP had a zAAP processor been available. It is a subset of the CP value. The system must be configured to capture this value. It is controlled by the parmlib option xxxxxxxxxxxxx. Our system did not have this option set. To convert this percentage to time is simple:
   
   \[(AAPCP \times \text{interval length}) / 100\]

7. **IIPCP** – same as AAPCP except for zIIP processors
8. **AAP** – this is the amount of zAAP time consumed during the interval. It reports the same value as the AAP field under the “SERVICE TIME” heading.
9. **IIP** – same as AAP except for zIIP processors.

The APPL% values are processor times reported as a percentage. They are reported as the percentage of a single processor so it is common to see values greater than 100% on multi-processor systems.

Given this information, calculating the amount of processor time used during the interval is very straightforward. The amount of zAAP processor time is simply the value reported in the AAP field, 2.015 seconds. Remember the CPU field contains the time spent on zAAPs so if we want to calculate the total amount of general purpose CP time we must subtract the AAP value from the total of the CPU and SRB values.
In the example above, which is a report class that defines enclave work, the SRB field will always be zero so to calculate the CP time we simply need to subtract the AAP value from the CPU value or 2.171 – 2.015 = 0.156. So in this example, an enclave service class, the total amount of CP and zAAP processor time spent by work executing under this report class is simply the CPU value.

Since we are using a WebSphere example we should also include the amount of processor time consumed by the deployment manager address spaces (control and servant), the node agent address space, and the application server address spaces (control and servant) (the SRB field is non-zero so remember to add that value to the CPU value to get the total amount of CP and zAAP time consumed during the interval. Then just subtract the AAP value from this total to get the amount of CP processor time.)

**Unix System Services (USS) and OMVS**

Display global USS settings: /D OMVS,O

```plaintext
BPX0043I 10.14.11 DISPLAY OMVS 616
OMVS 000F ACTIVE OMVS=(S4)
CURRENT UNIX CONFIGURATION SETTINGS:
MAXPROCSYS = 1900   MAXPROCUSER = 500
MAXFILEPROC = 65535  MAXFILESIZE = NOLIMIT
MAXCPU TIME = 2147483647  MAXUIDS = 500
MAXPTYS = 750
MAXMMAPAREA = 128K   MAXASSIZE = 2147483647
MAXTHREADS = 50000   MAXSHAREPAGES = 4M...
MAXCORESIZE = 7921K  MAXSHAREPAGES = 4M...
MAXQUEUEDSIGS = 10000  SHRLIBRGNSIZE = 67108864...
```

Other useful commands:

- `ps -p <pid_number> -o pid,ppid,xasid,thdcnt,vsz=VIRTUAL -o atime,jobname,ruser,etime \ -o args`
- `ps -p <pid_number> -o pid,xasid,xstid,xtcbaddr,wtime,semnum,lpid,lsyscall,syscall,atime,state \ -o THREAD -o comm=CMD`

**Language Environment (LE)**

z/OS provides a built-in mechanism to recommend fine tuned values for the LE heap. Run with LE RPTSTG(ON) and consult the resulting output:


Ensure that you are NOT using the following options during production: RPTSTG(ON), RPTOPTS(ON), HEAPCHK(ON)

For best performance, use the LPALSTxx parmlib member to ensure that LE and C++ runtimes are loaded into LPA.

Ensure that the Language Environment® data sets, SCEERUN and SCEERUN2, are authorized to enable xplink... For best performance, compile applications that use JNI services with xplink enabled.
FTP
FTP can be used to download both USS files as well as data sets. To download a data set, surround the
data set name with apostrophes:

```bash
ftp> ascii
200 Representation type is Ascii NonPrint
ftp> get ‘WITADM1.SPF1.LIST’
...
```

To convert character sets from EBCDIC to ASCII, use FTP ASCII mode. If the file was written on the
z/OS system with an ASCII character set, then download the file using FTP BINARY mode.

**Input/Output (I/O)**
Ensure that DASD are of the fastest speed, striping, etc.

**Networking**
To discover the host name, run the system command /D SYMBOLS and find the TCPIP address space
name. In the TCPIP address space joblogs output, find the TCPIP profile configuration data set:

```
PROFILE DD DISP=SHR,DSN=TCPIP.PROFILE(&SYSN.)...
```

In 3.4, browse this dataset and this will show the host name and IP address mapping.

Increase MAXSOCKETS and MAXFILEPROC to 64000
(http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tun
etcpip.html,
http://www.ibm.com/support/knowledgecenter/SSLTBW_2.2.0/com.ibm.zos.v2r2.bpxb200/mxflprc.htm)

Tune TCP/IP buffer sizes (TCPSENDFRSIZE=131070, TCPRCVBUFRSIZE=131070)
(http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tun
etcpip.html)

Consider disabling delayed acknowledgments (NODELAYACKS). Warning: This option may or may
not be better depending on the workload (see the discussion of delayed acknowledgments).
(http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tun
etcpip.html)

Set SOMAXCONN=511
(http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tun
etcpip.html)

Monitoring dispatch requests:
oc/ae/tprf_monitor_dispatch_requests.html
Type HOMETEST in ISPF COMMAND to get the IP hostname and address.

**Resource Recovery Service (RRS)**

RRS is used to guarantee transactional support.

For best throughput, use coupling facility (CF) logger for the RRS log.

Ensure that your CF logger configuration is optimal by using SMF 88 records.

Set adequate default values for the LOGR policy.

If you don't need the archive log, you should eliminate it since it can introduce extra DASD I/Os. The archive log contains the results of completed transactions. Normally, the archive log is not needed.


**SVCDUMPs, SYSTDUMPs**

Issue the following command to start dump processing:

```
/DUMP COMM='Dump Description'
83 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
```

You will use the number 83 (WTOR) in this case to reply to the system with dump parameters.

In order to reply to the system with the appropriate dump parameters, you need to know the address space ID of the address space you want to dump. There are other options for dumping address spaces; however, we are going to stick to 1 address space at a time using the method in this section. To find the ASIDX go to SDSF.DA (page right with F11).

The template for replying to a dump for a WebSphere address space:

```
[xx],ASID=(yyy),SDATA=(RGN,TRT,CSA,NUC,PSA,GRSQ,LPA,SQA,SUM)
```

The reply to dump the servant ASIDX 16D is as follows (in SDSF.LOG):

```
/R 83,ASID=(16D),SDATA=(RGN,TRT,CSA,NUC,PSA,GRSQ,LPA,SQA,SUM)
```

After 2 minutes or so the following appears:

```
IEF196I IEF285I SYS1.DUMP.D111011.T193242.S34.S00005 CATALOGED
IEF196I IEF285I VOL SER NOS= XDUMP8.
IEA611I COMPLETE DUMP ON SYS1.DUMP.D111011.T193242.S34.S00005 646
```

The "complete dump on" dataset can be downloaded in binary.

**svcdump.jar**

svcdump.jar is an "AS IS" utility that can process SVCDUMPs and print various information:
Examples:
- Print threads: java -cp svcdump.jar com.ibm.zebedee.dtfj.PrintThreads <dumpname>

**Security**

When a SAF (RACF® or equivalent) class is active, the number of profiles in a class will affect the overall performance of the check. Placing these profiles in a (RACLSTed) memory table will improve the performance of the access checks. Audit controls on access checks also affect performance. Usually, you audit failures and not successes.

Use a minimum number of EJBROLEs on methods.

If using Secure Sockets Layer (SSL), select the lowest level of encryption consistent with your security requirements. WebSphere Application Server enables you to select which cipher suites you use. The cipher suites dictate the encryption strength of the connection. The higher the encryption strength, the greater the impact on performance.

Use the RACLST to place into memory those items that can improve performance. Specifically, ensure that you RACLST (if used): CBIND, EJBROLE, SERVER, STARTED, FACILITY, SURROGAT

If you are a heavy SSL user, ensure that you have appropriate hardware, such as PCI crypto cards, to speed up the handshake process.

Here's how you define the BPX.SAFFASTPATH facility class profile. This profile allows you to bypass SAF calls which can be used to audit successful shared file system accesses.

Define the facility class profile to RACF.

RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE)

Activate this change by doing one of the following:
re-IPL
invoke the SETOMVS or SET OMVS operator commands.

Use VLF caching of the UIDs and GIDs

Do not enable global audit ALWAYS on the RACF (SAF) classes that control access to objects in the UNIX file system. If audit ALWAYS is specified in the SETR LOGOPTIONS for RACF classes DIRACC, DIRSRCH, FSOBJ or FSSEC, severe performance degradation occurs. If auditing is required, audit only failures using SETR LOGOPTIONS, and audit successes for only selected objects that require it. After changing the audit level on these classes, always verify that the change has not caused an unacceptable impact on
response times and/or CPU usage.


Global Resource Serialization (GRS)
Check global resource contention: /D GRS,C

ISG343I 16.57.02 GRS STATUS 300
NO ENQ RESOURCE CONTENTION EXISTS
NO REQUESTS PENDING FOR ISGLOCK STRUCTURE
NO LATCH CONTENTION EXISTS

WebSphere® Application Server for z/OS® uses global resource serialization (GRS) to communicate information between servers in a sysplex... WebSphere Application Server for z/OS uses GRS to determine where the transaction is running.

WebSphere Application Server for z/OS uses GRS enqueues in the following situations: Two-phase commit transactions involving more than one server, HTTP sessions in memory, Stateful EJBs, "Sticky" transactions to keep track of pseudo-conversational states.

If you are not in a sysplex, you should configure GRS=NONE, or if you are in a sysplex, you should configure GRS=STAR. This requires configuring GRS to use the coupling facility.


z/VM

Memory Overcommit

In this document, we will define [overcommit] as the total of the virtual memory of the started (logged on) virtual machines to the total real memory available to the z/VM system.

When planning whether memory can be overcommitted in a z/VM LPAR, the most important thing is to understand the usage pattern and characteristics of the applications, and to plan for the peak period of the day. This will allow you to plan the most effective strategy for utilizing your z/VM system’s ability to overcommit memory while meeting application-based business requirements.

For z/VM LPARs where all started guests are heavily-used production WAS servers that are constantly active, no overcommitment of memory should be attempted.
In other cases where started guests experience some idle time, overcommitment of memory is possible.


IBM i

IBM i product documentation: http://www-01.ibm.com/support/knowledgcenter/ssw_ibm_i/welcome

IBM Java on IBM i runs in PASE mode, so most of its behavior is the same as on AIX: http://www-01.ibm.com/support/knowledgcenter/ssw_ibm_i_72/rzalf/rzalfwhatispase.htm?lang=en

IBM i Recipe

1. **CPU core(s)** should not be consistently saturated.
2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.
9. Enable **Collection Services** for performance data.

Also review the general topics in the Operating Systems chapter.

Central Processing Unit (CPU)

IBM Systems Workload Estimator and processStats: https://www-912.ibm.com/estimator

WRKSYSSTS:

Use Collection Services performance data to gather detailed performance information:

IBM iDoctor for IBM i PEX Analyzer

- Simplifies the collection and enhances the analysis of all types of PEX data, which includes, PROFILE, STATS and TRACE data.
- Provides the details necessary for the low-level analysis of processor utilization, DASD
operations, file space usage, waits, file opens and much more.

**IBM iDoctor for IBM i Job Watcher**

- Provides real-time, non-intrusive, detailed and summarized views of job/thread/task performance data.
- It's the step to take to avoid a system wide trace or to ensure that a trace will yield useful data.
- It's a super WRKSYSACT that displays both "running" and "waiting" components for a job.


**OS CPU Profiling**

Profiling the CPU on the IBM i can be done on a global or individual job (JVM) basis. This is used with the Performance Explorer (PEX) tool. The process to gather the data is as follows:


   ```
   ADDPEXDFN DFN(JVMCPU) TYPE(*TRACE) JOB((ALL/ALL/JVMNAME *ALL))
   MAXSTG(1000000) INTERVAL(1) TRCTYPE(*SLTEVT) SLTEVT(*YES) BASEVT((PMCO *NONE
   *FORMAT2))
   ```

2. Gather the data using the above PEX definition:

   ```
   STRPEX SSNID(TRACE1) DFN(JVMCPU)
   ```

3. Wait 5-10 minutes while the JVM is using high CPU, and then end the collection:

   ```
   ENDPEX SSNID(TRACE1) DTALIB(QPEXDATA)
   ```

4. Print the PEX report, first by program, and next by statement:

   ```
   PRTPEXRPT MBR(TRACE1) LIB(QPEXDATA) TYPE(*PROFILE) PROFILEOPT(*SAMPLECOUNT
   *PROGRAM)
   PRTPEXRPT MBR(TRACE1) LIB(QPEXDATA) TYPE(*PROFILE) PROFILEOPT(*SAMPLECOUN T
   *STATEMENT)
   ```

5. This produces two spool files to show the breakout of CPU. Here is a histogram showing the breakdown:

<table>
<thead>
<tr>
<th>All Jobs/Tasks CPU</th>
<th>41212922</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jobs in Collection CPU</td>
<td>41212922</td>
</tr>
</tbody>
</table>
   | Job CPU                 | 38813410| 94.2 %
   | Task CPU                | 2399512 | 5.8 %

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Job/Task Name</th>
<th>Pool</th>
<th>Priority</th>
<th>Existence</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000000000008E1</td>
<td>WQLWI7</td>
<td>QWEBQRYADM</td>
<td>976015</td>
<td>2</td>
</tr>
<tr>
<td>211403580</td>
<td>37479915</td>
<td>90.94</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

   Cnt  %  %
### Per thread CPU usage

Gathering per thread CPU usage can be done in a variety of ways. The best is to use the WRKJVMJOB command. Example:

1. **WRKJVMJOB**

2. **Take option 11 to display threads.** This shows the total CPU (seconds) for each thread.

```plaintext
Job . . . . . . : TJH80EXP          PID . . . . . . : 82839
User . . . . . . : QEJBSVR           JDK . . . . . . : 1.6.0
Number . . . . . : 946396           Bits . . . . . . : 32

Type options, press Enter.
10=Display call stack
```

<table>
<thead>
<tr>
<th>Thread</th>
<th>Name</th>
<th>Status</th>
<th>Total</th>
<th>Aux</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000087</td>
<td>P=704863:0=0:CT</td>
<td>THDW</td>
<td>10.336</td>
<td>8277</td>
</tr>
<tr>
<td>0000008A</td>
<td>JIT Compilation &gt;</td>
<td>THDW</td>
<td>76.830</td>
<td>809</td>
</tr>
<tr>
<td>0000008B</td>
<td>JIT Compilation &gt;</td>
<td>THDW</td>
<td>67.357</td>
<td>6</td>
</tr>
<tr>
<td>0000008C</td>
<td>JIT Compilation &gt;</td>
<td>THDW</td>
<td>42.743</td>
<td>3</td>
</tr>
<tr>
<td>0000008E</td>
<td>IProfiler</td>
<td>THDW</td>
<td>4.275</td>
<td>0</td>
</tr>
<tr>
<td>0000008F</td>
<td>Signal Dispatch &gt;</td>
<td>THDW</td>
<td>64.984</td>
<td>0</td>
</tr>
</tbody>
</table>
Another option would be to take option 13 instead of 11. This produces a spool file that can be displayed and sent to support.

**Physical Memory (RAM)**


**Input/Output (I/O)**

WRKDSKSTS shows the status of the disk drives. Look for "hot" drives indicating high %Busy. Units consistently above 30% busy will have slow IO response times.

Work with Disk Status

```
RCHM199B 09/09/13
12:27:05
Elapsed time: 00:00:22
```

<table>
<thead>
<tr>
<th>Unit</th>
<th>Type</th>
<th>Size (M)</th>
<th>% Used</th>
<th>I/O Rqs</th>
<th>Request Size (K)</th>
<th>Read Rqs</th>
<th>Write Rqs</th>
<th>Read (K)</th>
<th>Write (K)</th>
<th>Busy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.0</td>
<td>4.0</td>
<td>.0</td>
<td>.0</td>
<td>4.0</td>
<td>.0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.3</td>
<td>4.5</td>
<td>.2</td>
<td>.0</td>
<td>4.0</td>
<td>6.0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.0</td>
<td>4.0</td>
<td>.0</td>
<td>.0</td>
<td>4.0</td>
<td>4.0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.0</td>
<td>4.0</td>
<td>.0</td>
<td>.0</td>
<td>4.0</td>
<td>4.0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.3</td>
<td>8.0</td>
<td>.1</td>
<td>.1</td>
<td>8.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.1</td>
<td>4.0</td>
<td>.0</td>
<td>.0</td>
<td>4.0</td>
<td>4.0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.1</td>
<td>4.0</td>
<td>.0</td>
<td>.0</td>
<td>4.0</td>
<td>4.0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>4327</td>
<td>61744</td>
<td>87.7</td>
<td>.1</td>
<td>4.0</td>
<td>.0</td>
<td>.1</td>
<td>4.0</td>
<td>4.0</td>
<td>0</td>
</tr>
</tbody>
</table>

F11 shows another view and the current status. Look for DEGRADED or FAILED units. This example shows they are all ACTIVE. No issues.

---Protection---

<table>
<thead>
<tr>
<th>Unit</th>
<th>ASP</th>
<th>Type</th>
<th>Status</th>
<th>Compression</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>DPY</td>
<td>ACTIVE</td>
<td></td>
</tr>
</tbody>
</table>
Networking

Tune TCP/IP buffer sizes. Use CHGTCPA to tune them up to 8096 KB:

Using Collection Services Performance Data

WAS provides scripts to enable collection services for performance data: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/tprf_collectionsservices.html

Gathering Javacores using WRKJVMJOB

Gathering javacores, core dumps, heap dumps, and JVM summary data is very simple on the IBM i. The WRKJVMJOB utility allows you to do all of this.

1. WRKJVMJOB

2. This produces a list of all the JVMs active on the system

   Work with JVM Jobs
   RCHM199B
   09/09/13
   12:11:42
   Active JVMs on system: 22
   Type options, press Enter.
   5=Work with   7=Display job log   8=Work with spooled files
   9=Display GC information   11=Display threads   12=Dump   13=Print
   Opt  Job Name    User       Number  Function          Status
   QSRVMON     QSYS        842707  JVM-ServiceMon     THDW
   QP0ZSPWT    HENDERAN    862730  JVM-WSPreLaunc     TIMW
   BENNIEDMGR  QEJBSVR     911766  PGM-jvmStartPa     THDW
   NODEAGENT  QEJBSVR     911778  PGM-jvmStartPa     THDW
   BENNIENODE  QEJBSVR     911779  PGM-jvmStartPa     THDW
   SHU85EONE   QEJBSVR     916849  PGM-jvmStartPa     THDW
   STIMSERVER  QEJBSVR     934284  PGM-jvmStartPa     THDW
   BENNIE      QEJBSVR     937798  PGM-jvmStartPa     THDW
   DMGR        QEJBSVR     941298  PGM-jvmStartPa     THDW
   12  TJH80EXP    QEJBSVR     946396  PGM-jvmStartPa     THDW

3. From this list, you can select option 12 to dump. By default, option 12 performs a javacore dump. To produce a different type of dump, you can select 12 next to the JVM, then hit F4 to prompt the command. This will allow you to change. Note the type of dump. (*JAVA = javacore, *SYSTEM = Core dump, *HEAP = heapdump.phd file is produced)

   Generate JVM Dump (GENJVMDMP)

   Type choices, press Enter.

   Job name . . . . . . . . . . . . . . . . . . . > TJH80EXP     Name
   User . . . . . . . . . . . . . . . . . . > QEJBSVR     Name
   Number . . . . . . . . . . . . . . . . . . . > 946396   000000-999999
   Type . . . . . . . . . . . . . . . . . . . > *JAVA  *JAVA, *SYSTEM, *HEAP
4. The dumps produced (javacore, heapdump, core dump) will be placed in the JVMs user home directory. The joblog for the JVM will show the location of the file. For example:

DSPJOBLOG JOB(946396/QEJBSVR/TJH80EXP)
JVMDUMP010I Java dump written to
/QIBM/UserData/WebSphere/AppServer/V8/Express/profiles/tjh80exp/javacore.2013
0909.121650.82839.0001.txt

JVM Monitoring

Viewing the Application Server JVM can be done through WRKACTJOB. This command shows the total CPU seconds, CPU %, and IO for the job based on the elapsed time:

```
RCHM199B
11:40:35
CPU %:     2.2     Elapsed time:   00:00:10     Active jobs:   339
```

Type options, press Enter.
2=Change 3=Hold 4=End 5=Work with 6=Release 7=Display message
8=Work with spooled files 13=Disconnect ...

```
--------Elapsed--------
Opt  Subsystem/Job  Type  Pool  Pty      CPU  Int    Rsp  AuxIO  CPU %
  QWAS8          SBS     2    0        .0                  0     .0
  TJH80EXP     BCH     2   20    3454.9                  0     .0
```

F11 shows further views, including number of threads, status, and function.

```
Opt  Subsystem/Job  User        Number  Type  CPU %  Threads
  QWAS8          QSYS        894103  SBS      .0        2
  TJH80EXP     QEJBSVR     946396  BCH      .0       74
```

Current
```
Opt  Subsystem/Job  User        Type  CPU %  Function        Status
  QWAS8          QSYS        SBS      .0                   DEQW
  TJH80EXP     QEJBSVR     BCH      .0  PGM-jvmStartPa   THDW
```

WRKSYSSTS shows the memory pool activity for the JVM. The WRKACTJOB above shows the WebSphere server "TJH80EXP" is running in system pool 2. The example output of WRKSYSSTS below shows system pool 2 as having 28,626MB allocated. The page faults are in faults/second, and split between DB and Non-DB faults. This is based on elapsed time.

```
WRKSYSSTS ASTLVL(*ADVANCED)
Work with System Status
RCHM199B
11:51:52
% CPU used . . . . . . . .: 2.0     System ASP . . . . . . . : 493.9
G
% DB capability . . . . : 0.0     % system ASP used . . . . .
```
Elapsed time . . . . . . : 00:07:58   Total aux stg . . . . . : 493.9
G
Jobs in system . . . . . : 3211   Current unprotect used . : 15970
M
% perm addresses . . . . . : .032   Maximum unprotect . . . : 22252
M
% temp addresses . . . . . : .569

Sys      Pool   Reserved    Max                                      Paging
Pool    Size M   Size M     Act  Pool        Subsystem   Library     Option
1    1187.55    606.00  +++++  *MACHINE                            *FIXED
2   28626.03     11.30    820  *BASE                               *CALC
3   13319.48       .56   1140  *INTERACT                           *CALC
4        .25       .00      5  *SPOOL                              *FIXED

The above shows very low page fault rate based on almost 8 minutes elapsed time. Also note the Wait-
Inel and Act-Inel counts as being 0. A higher value indicates the max act value is too low for the
amount of threads active in the pool. This would cause performance problems.

F11 again shows the pool names. System pool 2 is the *BASE pool. This is the default pool for IBM i
batch processes, including WebSphere

Windows

Windows Recipe

1. **CPU core(s)** should not be consistently saturated.
2. Generally, **physical memory** should never be saturated and the operating system should not page
   memory out to disk.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not
   have poor response times.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic
   effects on performance.
5. Consider changing Processor Performance Management (PPM) to the "High Performance"
   setting or disabling it.
6. Operating system level statistics and optionally process level statistics should be periodically
   monitored and saved for historical analysis.
7. Review operating system logs for any errors, warnings, or high volumes of messages.
8. Review snapshots of process activity, and for the largest users of resources, review per thread
activity.
9. If the operating system is running in a virtualized guest, review the configuration and whether
or not resource allotments are changing dynamically.
10. Use Perfmon to review performance activity.
11. Use the Windows Performance Toolkit to review sampled native processor usage.

Also review the general topics in the Operating Systems chapter.

General
Check the Windows Event log (eventvwr.exe) for any warnings, error messages, or repeated
informational messages.
Microsoft performance tuning guidelines by server version: https://msdn.microsoft.com/en-
us/library/windows/hardware/dn529134

Command Prompt
Recursive search for a file pattern:

```plaintext
> @echo off
> for /F "usebackq" %i in (`dir /s /b *.pdb`) do echo %i
> @echo on
```

Windows Registry
Many operating system settings are changed in the Windows registry. To open the registry, execute regedit.exe.
We recommend periodically backing up the registry, particularly before any significant changes:
- File > Export
- Export Range=All
- Save as some file.reg

Performance Monitor (Perfmon)
Perfmon is the generally recommended tool for Windows performance analysis.
"Windows Performance Monitor is a Microsoft Management Console (MMC) snap-in that provides
tools for analyzing system performance. From a single console, you can monitor application and
hardware performance in real time, customize what data you want to collect in logs, define thresholds
for alerts and automatic actions, generate reports, and view past performance data in a variety of ways."

By default, counters do not show the process ID, so with multiple java processes, they are java_N, and
if one process ends, all counters N+1 actually change. It is recommended to change to the PID format
(https://support.microsoft.com/kb/281884):

```plaintext
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\PerfProc\Performance
DWORD ProcessNameFormat=2
```
No restarts of the machine or Java are required - just restart Perfmon if it was open.

**View Live Data**

In the left pane, select Performance > Monitoring Tools > Performance Monitor.

This will show a live graph of % Processor Time:

![Performance Monitor Graph]

Some useful tips:

- To delete a counter, select the row in the bottom table and click Delete.
- Click the pencil toggle button to highlight the currently selected counter.
- By default, all counter values are scaled between 0 and 100. You can see if values are scaled by looking at the Scale column.
- "Last" is the last sampled value ("Minimum" and "Maximum" are also useful).
- "Average" is the average of all sampled values.
- "Duration" is the amount of time (rolling) that Perfmon will capture and display data. To extend this, right click on the graph > Properties > General > Duration = X seconds
- There are more options in the properties dialog that are worth exploring.

To add a counter, click the green plus icon:
Select a counter and the instances and click Add >>. In general, select <All instances> to ensure you get all the data. For example, if you select Process > % Processor time and you select <All instances>, if a process is spawned after data collection starts, it will be captured.

The instances are a way to look at counters in a more granular way. For example, the 0 and 1 instances above correspond to the two processors on this machine. If we select _Total, we will get the average of both processors. If we select <All instances>, this is a convenience and it is equivalent to multi-selecting _Total, 0, and 1.

Check "Show description" to better understand each counter.

**Logging Perfmon Data to Files**

For historical analysis or problem determination, configure Perfmon to log data to files:

1. Start Perfmon.exe.
2. Performance > Data Collector Sets
3. Right click User Defined > New > Data Collector Set
4. In the "Create new Data Collector Set" window, in the "Name:" field, type the name you would like, then click the "Create manually (Advanced)" button and click Next.

5. Under the "Create data logs" section, select the Performance counter box, and click Next.

6. In the "Performance counters:" section, click the "Add" button. Select the following counters:
   2. Memory > Available MBytes, Cache Bytes, Cache Faults/sec, Committed Bytes, Free System Page Table Entries, Page Faults/sec, Pages Input/sec, Pages Output/sec, Pool Nonpaged Bytes, Pool Pages Bytes, System Cache Resident Bytes
   3. Network Interface > Bytes Received/sec, Bytes Sent/sec, Output Queue Length, Packets Outbound Discarded, Packets Outbound Errors, Packets Received Discarded, Packets Received Errors > <All instances>
   4. Paging File > % Usage
   5. Process > % Privileged Time, % Processor Time, % User Time, IO Data Bytes/sec, IO Data Operations/sec, IO Other Bytes/sec, IO Other Operations/sec > <All instances>
   6. Processor > % Interrupt Time, % Privileged Time, % Processor Time, % User Time > <All instances>
   7. Processor Information > % Interrupt Time, % Privileged Time, % Processor Time, % User Time > <All instances>
   9. Thread > % Processor Time, ID Process, ID Thread > <All instances>

7. Change the "Sample Interval" to 30 seconds, click Next.

8. In the "Where would you like the data to be saved?" section, change the path for the Perfmon files if you would like, click Next.

9. In the "Create the data collector set?" section, click on the Finish button.

10. Ensure that the directory where the Perfmon files will be written has sufficient space.

11. Start the collection by right clicking and clicking Start.

12. The files saved will have a .blg extension.


**Load Existing Logs into Perfmon**

1. In the left pane, select Performance > Monitoring Tools > Performance Monitor.
2. Select the icon for View Log Data.
3. Select Log files: and click Add... and browse to the location of the Perfmon blg log files.
4. Click Add to select from the available counters in the data.

**typeperf**

The Windows typeperf command allows for simple access to performance counters from the command line: [https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/typeperf](https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/typeperf)
Central Processing Unit (CPU)

The key Perfmon counters are Process > % Interrupt Time, % Privileged Time, % Processor Time, % User Time > <All instances>. Note that the processor statistics for a particular process are in terms of a percentage of total CPU time, so if a process is using 2 CPUs at 100%, the sampled value will be 200.

"Where the "_Total" line reaches 100%, the Java process probably became constrained on CPU. If all the CPU is being used by the Java process, the performance is being limited by the machine. If another process is taking large amounts of CPU at those points in time, CPU contention is limiting the performance of the Java process." (Old Java Diagnostic Guide)

Per-Thread CPU Usage

With the Perfmon Thread counters, identify the threads that are using high CPU and convert the "ID Thread" value to hexadecimal. On IBM Java, if a thread dump was taken during these high CPU times, search the javacore file for the hexadecimal identifier to find the Java stack:

> The reason for generating per-thread CPU usage information about the Java process is to understand what is happening to the process. The Java process might be deadlocked if all the threads are taking little or no CPU time. Points of contention or delay might be in the Java process if it does not take all the available CPU, even though the CPU usage is spread evenly over a number of threads in the process. This CPU usage pattern might also indicate a scalability issue. Finally, you might have a looping problem if the Java CPU usage is approaching 100% and a small number of the threads account for all of that CPU usage. The threads using the most process time might be looping. When you find some threads of interest, note the ID Thread values. Convert the values to hexadecimal, and look for the threads in the thread stack trace of the javacore.txt file. This trace helps you to determine if the thread is part of a thread pool and to understand what kind of work the thread performs. For example, an ID thread of 9244 becomes 241C in hexadecimal and is found in the "native ID" value in the javacore.txt file.

Perfmon counters: "% Processor Time", "ID Thread", and any other counters in which you are interested for all the Java thread instances

Old Java Diagnostic Guide

PsList


In most modes, you can filter the results by passing a process name prefix (such as java) or a PID at the end of the command.

No arguments prints the accumulated CPU time of each process and the elapsed time each process has been running:

> pslist.exe
Process information for ADMINIB-I6CU78U:

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>Pri</th>
<th>Thd</th>
<th>Hnd</th>
<th>Priv</th>
<th>CPU Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>11:08:07.609</td>
<td>0:00:00.000</td>
</tr>
<tr>
<td>System</td>
<td>4</td>
<td>8</td>
<td>82</td>
<td>4062</td>
<td>108</td>
<td>0:01:36.500</td>
<td>5:41:15.690</td>
</tr>
<tr>
<td>smss</td>
<td>240</td>
<td>11</td>
<td>2</td>
<td>30</td>
<td>440</td>
<td>0:00:01.484</td>
<td>5:41:13.940</td>
</tr>
<tr>
<td>csss</td>
<td>316</td>
<td>13</td>
<td>9</td>
<td>871</td>
<td>2324</td>
<td>0:00:02.312</td>
<td></td>
</tr>
</tbody>
</table>

5:40:51.518...

The pslist argument -s shows an auto-updating view similar to task manager (similar to the top command on Unix platforms):

```
> pslist -s
2:24:04 PM 2/5/2014 Process information for ADMINIB-I6CU78U:

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>CPU</th>
<th>Thd</th>
<th>Hnd</th>
<th>Priv</th>
<th>CPU Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>97</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>11:15:27.906</td>
<td>5:45:06.985</td>
</tr>
<tr>
<td>pslist</td>
<td>4348</td>
<td>3</td>
<td>2</td>
<td>155</td>
<td>2840</td>
<td>0:00:02.015</td>
<td>0:00:30.546</td>
</tr>
<tr>
<td>smss</td>
<td>240</td>
<td>0</td>
<td>2</td>
<td>30</td>
<td>440</td>
<td>0:00:01.484</td>
<td></td>
</tr>
<tr>
<td>csss</td>
<td>316</td>
<td>0</td>
<td>9</td>
<td>847</td>
<td>2324</td>
<td>0:00:02.312</td>
<td></td>
</tr>
<tr>
<td>java</td>
<td>2684</td>
<td>8</td>
<td>87</td>
<td>589</td>
<td>315196</td>
<td>3:15:19.663</td>
<td></td>
</tr>
<tr>
<td>csss</td>
<td>364</td>
<td>0</td>
<td>8</td>
<td>403</td>
<td>2504</td>
<td>0:00:01.234</td>
<td></td>
</tr>
<tr>
<td>wininit</td>
<td>372</td>
<td>0</td>
<td>3</td>
<td>173</td>
<td>2728</td>
<td>0:00:00.265</td>
<td></td>
</tr>
<tr>
<td>winlogon</td>
<td>404</td>
<td>0</td>
<td>3</td>
<td>113</td>
<td>2728</td>
<td>0:00:00.265</td>
<td></td>
</tr>
</tbody>
</table>

5:44:41.188...

The pslist argument -t shows a tree view of process ownership:

```
> pslist -t
```

Process information for ADMINIB-I6CU78U:

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>Pri</th>
<th>Thd</th>
<th>Hnd</th>
<th>VM</th>
<th>WS</th>
<th>Priv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>24</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>System</td>
<td>4</td>
<td>8</td>
<td>82</td>
<td>4030</td>
<td>3380</td>
<td>300</td>
<td>108</td>
</tr>
<tr>
<td>smss</td>
<td>240</td>
<td>11</td>
<td>2</td>
<td>30</td>
<td>4024</td>
<td>1100</td>
<td>440</td>
</tr>
<tr>
<td>java</td>
<td>2684</td>
<td>8</td>
<td>87</td>
<td>589</td>
<td>315196</td>
<td>3:15:19.663</td>
<td></td>
</tr>
<tr>
<td>csss</td>
<td>316</td>
<td>13</td>
<td>9</td>
<td>583</td>
<td>50260</td>
<td>4780</td>
<td>2324</td>
</tr>
<tr>
<td>csss</td>
<td>364</td>
<td>13</td>
<td>8</td>
<td>406</td>
<td>210896</td>
<td>12332</td>
<td>2504</td>
</tr>
<tr>
<td>conhost</td>
<td>3484</td>
<td>8</td>
<td>2</td>
<td>79</td>
<td>77380</td>
<td>9916</td>
<td>4400</td>
</tr>
<tr>
<td>wininit</td>
<td>372</td>
<td>13</td>
<td>3</td>
<td>77</td>
<td>321196</td>
<td>4302</td>
<td>1472</td>
</tr>
<tr>
<td>services</td>
<td>400</td>
<td>9</td>
<td>9</td>
<td>248</td>
<td>45168</td>
<td>9796</td>
<td>6204</td>
</tr>
<tr>
<td>svchost</td>
<td>580</td>
<td>8</td>
<td>10</td>
<td>362</td>
<td>46512</td>
<td>9492</td>
<td>3832</td>
</tr>
<tr>
<td>WmiPrvSE</td>
<td>2152</td>
<td>8</td>
<td>7</td>
<td>339</td>
<td>80312</td>
<td>16304</td>
<td>8368</td>
</tr>
<tr>
<td>ProtectionUtilSurrogate</td>
<td>4036</td>
<td>8</td>
<td>10</td>
<td>289</td>
<td>98168</td>
<td>13184</td>
<td>4304...</td>
</tr>
</tbody>
</table>

The pslist argument -d prints the accumulated CPU times of each thread as well as the elapsed times the threads have existed:

```
> pslist -d java
```

Thread detail for ADMINIB-I6CU78U:

```
java 2684:

<table>
<thead>
<tr>
<th>Tid</th>
<th>Pri</th>
<th>Cswtch</th>
<th>State</th>
<th>User Time</th>
<th>Kernel Time</th>
<th>Elapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The `pslist` argument `-x` is the same as `-d` but also prints memory information about the process (to get processes' memory without threads, use `-m`):

```
> pslist -x java
```

**Process and thread information for ADMINIB-I6CU78U:**

<table>
<thead>
<tr>
<th>Tid</th>
<th>Pri</th>
<th>Cswtch</th>
<th>State</th>
<th>User Time</th>
<th>Kernel Time</th>
<th>Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2688</td>
<td>9</td>
<td>6</td>
<td>Wait:UserReq</td>
<td>0:00:00.000</td>
<td>0:00:00.000</td>
<td>5:47:24.155</td>
</tr>
<tr>
<td>2696</td>
<td>9</td>
<td>8465</td>
<td>Wait:UserReq</td>
<td>0:00:07.515</td>
<td>0:00:06.906</td>
<td>5:47:24.155</td>
</tr>
<tr>
<td>2700</td>
<td>8</td>
<td>22</td>
<td>Wait:UserReq</td>
<td>0:00:00.000</td>
<td>0:00:00.000</td>
<td>5:47:24.155</td>
</tr>
<tr>
<td>2704</td>
<td>15</td>
<td>8401</td>
<td>Wait:UserReq</td>
<td>0:00:08.921</td>
<td>0:00:02.203</td>
<td>5:47:24.092</td>
</tr>
<tr>
<td>2716</td>
<td>15</td>
<td>1146643</td>
<td>Wait:UserReq</td>
<td>0:00:00.000</td>
<td>0:00:00.000</td>
<td>5:47:23.733</td>
</tr>
<tr>
<td>2720</td>
<td>9</td>
<td>33519</td>
<td>Wait:UserReq</td>
<td>0:00:00.578</td>
<td>0:00:00.468</td>
<td>5:47:22.733</td>
</tr>
</tbody>
</table>

**Windows Performance Toolkit (WPT)**

The Windows Performance Toolkit is a free tool from Microsoft that provides various dimensions of performance analysis: https://msdn.microsoft.com/en-us/library/windows/hardware/hh162945.aspx

**Installation**

2. On the "Select the features you want to install" screen, only "Windows Performance Toolkit" is required.
3. On 64-bit Windows 7 and Windows Server 2008 (but not newer versions such as Windows 8 and Windows Server 2012), add the following registry entry and reboot:

```
> REG ADD "HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management" -v DisablePagingExecutive -d 0x1 -t REG_DWORD -f
```

**Collect Data**

There are two main ways to collect data (ETL file):

1. Run C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\WPRUI.exe
   1. Leave the defaults of Performance Scenario=General, Detail level=Verbose, Logging mode=Memory
   2. Check Resource Analysis > CPU usage
3. Click Start, Reproduce the problem, Click Save
2. Run C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\xperf.exe from the command line.
   1. Start collection:
   2. > xperf -on SysProf -stackwalk Profile -BufferSize 1024 -MaxBuffers 320
   3. These options buffer data to memory, so significant additional RAM may be needed.
      There are also options to flush to files.
   4. Stop collection:
   5. > xperf -d callstacks.etl

By default, WPT data is written to %HOMEPATH%\Documents\WPR Files\*.etl. When clicking the
"Start" button, the old collection files are not overwritten.

Analyze Data
There are three main ways to view an ETL file:

1. Windows Performance Analyzer (WPA.exe %ETL%)
2. Trace > Configure Symbol Paths
   If .NET code was running at the time of the capture, an NGENPDB folder will be automatically
   created under %HOMEPATH%\Documents\WPR Files\ with the name of the .etl file. If it may
   be necessary to investigate .NET code, copy this path, which is automatically included in the
   default symbol path in WPA, and add to the end of the final symbol path.

   Example:
   C:\work\WAS8554_20140924\java\jre\bin;C:\work\WAS8554_20140924\java\jre\bin\compresedrefs;C:\work\WAS8554_20140924\lib\native\win\x86_64;\srv*C:\Symbols*http://msdl.microsoft.com/download/symbols;

3. Trace > Load Symbols

Absolute Times
WPA shows all data in relative terms (seconds). Unfortunately, there doesn’t appear to be an option to
use absolute timestamps. To determine when the tracing started:

1. Click Trace > System Configuration
2. Click Traces
3. Review Start Time (UTC)

The default ETL file name will include the date and time in local format, but this appears to be roughly
the time the trace is requested to be stopped.

It is common for a ~200 second delay between the start of the capture and availability of some data
(presumably while the kernel is initializing tracing).

CPU Analysis

1. Expand Computation > CPU Usage (Sampled) > Utilization by Process, Stack
Flamegraphs can also be generated: https://randomascii.wordpress.com/2013/03/26/summarizing-xperf-cpu-usage-with-flame-graphs/

CPU Usage by Thread

1. Expand Computation > CPU Usage (Attributed) > Utilization by Process, Thread, Activity *
Disk Analysis

1. Expand Storage > Disk Usage > Service Time by Process, Path Name, Stack
2. The disk times are in microseconds

Analyzing on Another Machine

1. Gather the etl file from %HOMEPATH%\Documents\WPR Files\n2. Gather all *.pdb files from the WebSphere folder.
3. If .NET code was running at the time of the capture, an NGENPDB folder will be automatically
   created under %HOMEPATH%\Documents\WPR Files\ with the name of the .etl file. If it may
   be necessary to investigate .NET code, also gather this folder.

TPROF

The open source performance inspector suite (originally created by IBM) includes a native Windows
sampling profiler called TPROF: http://perfinsp.sourceforge.net/tpprof.html

This is a great way to understand which native modules are using the CPU and it is requested as part of
the IBM Performance MustGather on Windows: http://www-01.ibm.com/support/docview.wss?uid=swg21111364

The reason this tool is so useful is that it is a sampling profiler (see the Java Profilers chapter for
background on this topic). It will sample the native stacks of the processes approximately every 7
milliseconds. This tends to be a very low overhead (less than a few percent) way to get insight into
CPU usage without dramatically impacting the system. In general, TPROF can be used in production environments, although you should fully test this in a test environment first.

The instructions to install and use TPROF are quite straightforward: [http://www-01.ibm.com/support/docview.wss?uid=swg21403450](http://www-01.ibm.com/support/docview.wss?uid=swg21403450)

Currently, TPROF does not work on Windows Server >= 2012.

Install with tinstall.cmd

Run with:

```bash
> setrunenv.cmd
> run.tprof.cmd
```

Press ENTER to start capturing data
Reproduce the problem
Press ENTER again to stop capturing data
Open tprof.out to analyze the results (see the TechNote above for a description of the various sections)

For example, in one case we were led to investigate some third party drivers by seeing a significant amount of CPU usage in the kernel (and other modules that are not shown here for confidentiality):

```
PID 695 51.00 java.exe_0c8c
MOD 320 20.46 C:\Windows\system32\ntoskrnl.exe
```

**Processor Performance Management (PPM)**

Processor Performance Management (PPM) is a power saving feature. It may be changed to the "High Performance" setting: [https://technet.microsoft.com/en-us/library/dd744398%28v=ws.10%29.aspx](https://technet.microsoft.microsoft.com/en-us/library/dd744398%28v=ws.10%29.aspx)

A common symptom in profilers such as TPROF is a high CPU usage in, for example, the intelppm.sys driver:

```
LAB  TKS  %%%  NAMES
MOD 20448  7.13  C:\Windows\system32\DRIVERS\intelppm.sys
```

For example, the intelppm driver may be disabled with the following command followed by a restart:

```
> sc config intelppm start= disabled
```

**Physical Memory (RAM)**


- Memory\Available bytes = The amount of free physical memory available for running processes.
- Memory\Cache bytes = The amount of physical memory used by the file system cache.
- Memory\Free System Page Table Entries = The number of free PTEs. Should be non-zero.
- Memory\Pool Non-Paged Bytes = Memory used by the kernel which cannot be paged out.
- Memory\Pool Paged Bytes = Memory used by the kernel which can be paged out.
Process Memory Usage


Windows 32-bit uses a default virtual user address space of 2GB ([http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/dw3gbswitch3.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/dw3gbswitch3.pdf)). This can be changed to a 3GB virtual user address space:


In 3GB mode, some libraries are still based at the 2GB boundary, so -Xmx is practically limited to between -Xmx1408m and -Xmx1856m because it is a single, contiguous allocation. Library rebasing is possible but then shared libraries are loaded privately.

Starting in IBM Java 6, the split heap option may be used which forces gencon and allows you to straddle nursery and tenured regions around the 2GB area. For example: -Xgc:splitheap -Xmx2800m -Xmox1800m ([http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.win.80.doc/diag/appendixes/cmdline/xgcsplitheap.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.win.80.doc/diag/appendixes/cmdline/xgcsplitheap.html))
A program must be linked with /LARGEADDRESSAWARE to utilize a system configured in a way other than the default 2GB mode. IBM Java is linked with this option.

"If an application was linked with /LARGEADDRESSAWARE, DUMPBIN /HEADERS will display information to that effect."


This option is not risk free: Third party JNI libraries with pointer arithmetic may have unexpected issues or crashes. The kernel itself may also run into issues, particularly with exhausted page translation table entries or an exhausted non-paged pool when there is a lot of network activity.

**Input/Output (I/O)**


- LogicalDisk\Avg. Disk sec/Read: Average time, in seconds, of a read of data from the disk
- LogicalDisk\Avg. Disk sec/Write: Average time, in seconds, of a write of data to the disk
- LogicalDisk\Current Disk Queue Length: Indicates the number of disk requests that are currently waiting as well as requests currently being serviced.
- LogicalDisk\%Idle Time: Reports the percentage of time that the disk system was not processing requests and no work was queued.
- LogicalDisk\Disk Reads/sec
- LogicalDisk\Disk Writes/sec
- LogicalDisk\Disk Read Bytes/sec
- LogicalDisk\Disk Write Bytes/sec
- Process\IO Read Bytes/sec
- Process\IO Write Bytes/sec

**Defragmentation**

As you delete files, you create gaps in the arrangement of the contiguously stored files. As you save new files (and this is especially true for large files), the file system uses up all of these bits of free space - resulting in the new files being scattered all over the disk in noncontiguous pieces. And thus we end up with fragmented disks and system performance issues because the disk heads have to spend time moving from cluster to cluster before they can read or write the data.

[The Disk Defragmenter] utility physically rearranges the files so that they are stored (as much as possible) in physically contiguous clusters. In addition to the consolidation of files and folders, the Defragmenter utility also consolidates free space - meaning that it is less likely for new files to be fragmented when you save them. For operating systems prior to Windows Vista, you had to manually run the utility or schedule automatic defragmentation via a scheduled task. On Windows Vista, Disk Defragmenter runs as a low-priority background task that is automatically run on a weekly basis without requiring user intervention. On Windows Server 2008, which uses the same Disk Defragmenter, the automatic defragmentation is not enabled by default... The basic operation of the utility involves passing it a driver letter, for example: defrag.exe c: would perform a defragmentation of the C: drive.
> defrag c: -a


CIFS/SMB
The most common protocols for a networked file systems on Windows are Common Internet File System (CIFS) and Server Message Block (SMB). The SMB version 2 protocol is new and no longer synonymous with CIFS (https://msdn.microsoft.com/en-us/library/ee441790.aspx).


If acceptable from a security point of view, consider disabling SMB packet signing: "By default, client-side SMB signing is enabled on workstations, servers, and domain controllers... Using SMB packet signing can degrade performance up to 15 percent on file service transactions" (https://technet.microsoft.com/en-us/library/cc731957.aspx) and "... the overhead could get extremely high—up to 40 percent in some situations" (https://technet.microsoft.com/en-us/library/cc512612.aspx). Disable 'Microsoft network client: Digitally sign communications (if server agrees)' and 'Microsoft network client: Digitally sign communications (always)'.

SMB2.1 introduces large Maximum Transmission Unit (MTU) support up to 1MB (https://technet.microsoft.com/en-us/library/ff625695(v=ws.10).aspx). It is enabled with HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\LanmanWorkstation\Parameters\DisableLargeMtu = 0 followed by a reboot (http://download.microsoft.com/download/9/B/2/9B205446-37EE-4BB1-9A50-E872565692F1/PerfTuningGuideServer2012R2.pdf).


Test the response time of an SMB copy using a large file by creating a batch file such as largefilecopy.bat:

```batch
@echo off
echo %TIME%
xcopy /J /Y %PATHTOLOCALFILE% \%SMBPATH%
echo %TIME%
```

One technique of determining what proportion of time a process spends waiting for SMB responses is to gather network trace, filter to the times spanning a particular process request, add a Wireshark column for smb2.time, export to CSV, sum the service response times, and compare to the elapsed time of the process request.

Some people suggest disabling "Domain member: Digitally encrypt secure channel data;" however, this option does not appear to be related to SMB traffic (https://technet.microsoft.com/en-us/library/jj852270(v=ws.10).aspx).
Networking

Update TIME_WAIT timeout:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpTimedWaitDelay = REG_DWORD value 30

Update maximum ephemeral local port:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\MaxUserPort = REG_DWORD value 65534

Consider disabling delayed TCP acknowledgments:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\TcpAckFrequency = REG_DWORD value 1. Warning: This option may or may not be better depending on the workload (see the discussion of delayed acknowledgments).

Consider increasing the TCP maximum window size. For example, to set the value to 65535,
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\GlobalMaxTcpWindowSize = REG_DWORD value 0xFFFF:

Consider increasing the maximum number of TCP/IP control blocks (MaxFreeTcbs) when using large numbers of connections: https://technet.microsoft.com/en-us/library/cc938178.aspx. When modifying MaxFreeTcbs, MaxHashTableSize must also be modified proportionally:

Starting with Windows Server 2008, it is no longer applicable to modify EnableDynamicBacklog/MinimumDynamicBacklog/MaximumDynamicBacklog/DynamicBacklogGrowthDelta

Increase network adapter receive buffers: https://support.microsoft.com/kb/981482

It appears that TCP/IP in Windows 2012 is the same as 2008, so all of the same tuning applies: "In Windows Server 2012, TCP/IP – including both Internet Protocol version 4 (IPv4) and IPv6 – is unchanged from TCP/IP in Windows Server 2008 R2. For more information, see TCP/IP in the Windows Server® 2008 and Windows Server 2008 R2 Technical Library."

Ping a remote host. In general, and particularly for LANs, ping times should be less than a few hundred milliseconds with little standard deviation.
> ping -t 10.20.30.1

Pinging 10.20.30.1 [10.20.30.1] with 32 bytes of data:
Reply from 10.20.30.1: bytes=32 time=92ms TTL=249
Reply from 10.20.30.1: bytes=32 time=89ms TTL=249
Reply from 10.20.30.1: bytes=32 time=155ms TTL=249
Reply from 10.20.30.1: bytes=32 time=89ms TTL=249

Ping statistics for 
10.20.30.1  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 89ms, Maximum = 155ms, Average = 106ms

The initial congestion window size may be changed with
(https://support.microsoft.com/kb/2472264):

> netsh interface tcp set supplemental template=custom icw=10
> netsh interface tcp set supplemental template=custom

netstat

Create a snapshot of socket information:

> netstat -a -b -n -o

Active Connections

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0.0.0.0:7278</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>2684</td>
</tr>
<tr>
<td>[java.exe]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:8881</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>2684</td>
</tr>
<tr>
<td>[java.exe]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>0.0.0.0:9045</td>
<td>0.0.0.0:0</td>
<td>LISTENING</td>
<td>2684</td>
</tr>
<tr>
<td>[java.exe]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Show adapter statistics:

C:\tprof\bin>netstat -s

IPv4 Statistics

Received Header Errors  = 0
Received Address Errors = 0
Unknown Protocols Received = 0
Received Packets Discarded = 9
Routing Discards = 0
Discarded Output Packets = 17
Output Packet No Route = 0
Reassembly Required = 0
Reassembly Failures = 0
Datagrams Failing Fragmentation = 0...

TCP Statistics for IPv4

Failed Connection Attempts  = 445
Reset Connections                     = 149
Segments Retransmitted               = 921...

Show ethernet statistics:
> netstat -e

Interface Statistics

<table>
<thead>
<tr>
<th></th>
<th>Received</th>
<th>Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>275244337</td>
<td>12757159...</td>
</tr>
<tr>
<td>Discards</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Errors</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Unknown</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Wireshark**

Capture network packets using Wireshark (covered in the [Major Tools chapter](#)).

Start the capture:

1. Install Wireshark: [https://www.wireshark.org/#download](https://www.wireshark.org/#download)
2. Start Wireshark as Administrator
3. Click "Capture" > "Options"
4. Select the network interface in the "Input" box
5. Click the "Output" tab and enter a "File" such as C:\wireshark.pcap
6. Click the "Options" tab and uncheck "Update list of packets in realtime" and click "Start"

Stop the capture:

1. Click "Capture" > "Stop"

**Message Analyzer**


**Network Monitor**

The official way to capture network packets on older versions of Microsoft Windows is Microsoft Network Monitor: [https://support.microsoft.com/kb/148942](https://support.microsoft.com/kb/148942)

**Process Monitor (ProcMon.exe)**


1. Delete any existing PML files from previous runs.
2. Command Prompt> ProcMon.exe /NoConnect (the /NoConnect option avoids immediately starting collection so that you can configure whatever's needed)
3. File > Backing Files > Select "Use file named" and enter a path such as C:\ProcMon.pml and click OK.
4. Filter > Uncheck "Drop Filtered Events"
5. Options > Configure Symbols... > Ensure DbgHelp.dll points to an existing path (install Debugging Tools if not), and set symbol paths to include a local symbol cache directory, such as srv*c:\symbols*http://msdl.microsoft.com/download/symbols
6. Options > Profiling Events > Check "Generate thread profiling events" and select "Every 100 milliseconds"
7. In the menu bar on the right, uncheck the 5 boxes named "Show Registry Activity, "Show File System Activity," etc. so that only the backing file is capturing the events and not the GUI as well.
8. File > Click Capture Events.
9. Reproduce problem
10. File > Uncheck "Capture Events" (or run ProcMon.exe /terminate from another command prompt). This step is required; otherwise, you may receive the following error when trying to open the PML files: "The file %FILE% was not closed cleanly during capture and is corrupt."
11. Load the PML File

**Thread Profiling Analysis**

Click Tools > Stack Summary..., sort by Time %, and expand the largest stack paths:

<table>
<thead>
<tr>
<th>Name</th>
<th>Count</th>
<th>% Count</th>
<th>Time</th>
<th>% Time</th>
<th>Location</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>U &lt;All&gt;</td>
<td>42817</td>
<td>100.00000%</td>
<td>29.57812...</td>
<td>100.00000%</td>
<td>U &lt;All&gt;</td>
<td>&lt;All&gt;</td>
</tr>
<tr>
<td>U java.exe</td>
<td>513</td>
<td>1.19812%</td>
<td>24.07812...</td>
<td>100.00000%</td>
<td>java.exe</td>
<td>java.exe(2812)</td>
</tr>
<tr>
<td>K K⇋pcInterrupt + 0x7</td>
<td>244</td>
<td>0.56987%</td>
<td>24,07812...</td>
<td>81,40518%</td>
<td>K⇋pcInterrupt + 0x7</td>
<td>ntoskrnl.exe</td>
</tr>
<tr>
<td>K K⇋DeliverApC + 0x166</td>
<td>244</td>
<td>0.56987%</td>
<td>24.07812...</td>
<td>81.40518%</td>
<td>K⇋DeliverApC + 0x166</td>
<td>ntoskrnl.exe</td>
</tr>
<tr>
<td>U RLIUserThreadSlat + 0x1d</td>
<td>244</td>
<td>0.56987%</td>
<td>0.0000000</td>
<td>0.00000%</td>
<td>RLIUserThreadSlat + 0x1d</td>
<td>ntoskrnl.dll</td>
</tr>
</tbody>
</table>

**Large Pages**

The -Xlp option requests the JVM to allocate the Java heap with large pages. This command is available only on Windows Server 2003, Windows Vista, Windows Server 2008, and above. To use large pages, the user that runs Java must have the authority to "lock pages in memory".

To enable this authority, as administrator go to Control Panel > Administrative Tools > Local Security Policy and then find Local Policies > User Rights Assignment > Lock pages in memory. Add the user who runs the Java process, and reboot your machine. For more information, see these websites:


Note: On Microsoft Windows Vista and Windows 2008, use of large pages is affected by the User Account Control (UAC) feature. When UAC is enabled, a regular user (a member of the Users group) can use the -Xlp option as normal. However, an administrative user (a member of the administrators group) must run the application as an administrator to gain the privileges required to lock pages in memory. To run as administrator, right-click the application and select Run as administrator. If the user does not have the necessary privileges, an error message is produced, advising that the System configuration does not support option '-Xlp'.
Solaris

Solaris Recipe

1. CPU core(s) should not be consistently saturated.
2. Program memory should not page out of RAM.
3. Input/Output interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. TCP/IP and network tuning, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.

Also review the general topics in the Operating Systems chapter.

General

Check the system log for any warnings, errors, or repeated informational messages.

    # less /var/adm/messages

Query the help manual for a command:

    $ man vmstat # By default, contents are sent to more
    $ man -a malloc # There may be multiple manuals matching the name. Use -a to show all of them.


Review the Solaris tuning in the latest SPECjEnterprise results submitted by Oracle:


The Solaris Management Console (smc) is no longer supported in recent releases:
http://docs.oracle.com/cd/E26502_01/html/E29010/gltfb.html

Processes

Query basic process information:

    $ ps -elf | grep java
By default, the process ID (PID) is the number in the fourth column. You can control which columns are printed and in which order using \texttt{-o}.

The built-in `ps` command may not show the entire command line. An alternative `ps` is often available:

\begin{verbatim}
$ /usr/ucb/ps auxwww
\end{verbatim}

\section*{Central Processing Unit (CPU)}

Query physical processor layout:

\begin{verbatim}
# psrinfo -pv
The physical processor has 16 cores and 128 virtual processors (0-127)
The core has 8 virtual processors (0-7)...
\end{verbatim}

\begin{verbatim}
# prtdiag -v
Memory size: 2GB
CPU Freq Size Implementation Mask Status
Location
0 1503 MHz 1MB SUNW,UltraSPARC-IIIi 3.4 on-line
MB/P0
1 1503 MHz 1MB SUNW,UltraSPARC-IIIi 3.4 on-line
MB/P1...
\end{verbatim}

Ensure there are no errant processes using non-trivial amounts of CPU.

\section*{vmstat}

Query processor usage:

\begin{verbatim}
$ vmstat 5 2
kthr memory page disk faults cpu
r b w swap free re mf pi po fr de sr s3 s5 s7 -- in sy cs us sy
id
0 0 0 4415400 739680 77 859 5 3 4 0 8 -0 3 -1 0 325 1634 476 2 2
96
0 0 0 4645936 1232224 3 5 0 0 0 0 0 0 285 349 274 0 1
99
\end{verbatim}

The documentation on the first line of \texttt{vmstat} is unclear:

Without options, \texttt{vmstat} displays a one-line summary of the virtual memory activity since the system was booted. (http://docs.oracle.com/cd/E19683-01/816-0211/6m6nc67ac/index.html)

Experimentation shows that, with options (such as interval or count), the first line also displays statistics since the system was booted:
# vmstat
kthr
memory
page
r b w
swap free re mf pi po
id
0 0 0 3932200 329624 79 857 1 1
95
# vmstat 5
kthr
memory
page
r b w
swap free re mf pi po
id
0 0 0 3932184 329616 79 857 1 1
95
0 0 0 3527808 70608 2780 25799 3
41
0 0 0 3527784 70728 2803 26009 0
42

disk
fr de sr s3 s5 s7 -1

0

2 -0

3 -0

0

disk
fr de sr s3 s5 s7 -1

0

faults
in
sy
351 1970
faults
in
sy

cpu
cs us sy
764

2

3

cpu
cs us sy

2 -0

3 -0

0

351 1970

764

2

3

2 2 0

0

0

2

0

0

445 14699 2383 15 44

0 0 0

0

0

0

0

0

430 14772 2387 15 44

Example to capture vmstat in the background:
INTERVAL=1; FILE=vmstat_`hostname`_`date +"%Y%m%d_%H%M"`.txt; date > ${FILE}
&& echo "VMSTAT_INTERVAL = ${INTERVAL}" >> $FILE && nohup vmstat ${INTERVAL}
>> $FILE &

Per processor utilization
Query per-processor utilization:
$ mpstat 5 2
CPU minf mjf xcal
0 425
0 115
1 434
0
98
CPU minf mjf xcal
0
0
0
1
1
2
0
3
99...

intr ithr
34
26
290 185
intr ithr
15
9
280 175

csw icsw migr smtx
202
7
51
14
274
5
52
16
csw icsw migr smtx
93
3
21
0
181
2
22
0

srw syscl
0
838
0
797
srw syscl
0
159
0
172

usr sys
2
2
2
2
usr sys
0
0
0
0

wt idl
0 96
0 96
wt idl
0 100
0

pgstat
pgstat: http://docs.oracle.com/cd/E23824_01/html/821-1462/pgstat-1m.html
prstat
By default, prstat prints the damped average % CPU statistics for processor usage by individual
processes or threads. Without arguments, prstat will periodically update the screen with relatively
accurate 'average' information (this may be at variance with data returned from vmstat due to the
difference in how it's calculated):
$ prstat

Although the prstat documentation does not explicitly mention this, by default, the reported CPU usage
is decayed over time. This can be confirmed with the Java program at
Page 165


For example, if a Java program uses 50% CPU from time T1 to time T2 (after which its CPU usage goes to approximately 0), and you start to take prstat at time T2, the first iteration will report about 50%, and the second iteration may report a decayed value, and so on in the following iterations. Therefore, prstat may not show the "current" processor usage of processes but may include some historical processor usage.

Use the -mv options to gather accurate interval-based statistics:

```
$ prstat -mv
```

For example, use prstat in micro-stat mode with the following options -mv for detailed, interval-accurate statistics, -n to limit the number of processes to report, and an interval and iteration count to print in batch mode:

```
$ prstat -mvcn ${MAXPROCESSES} ${INTERVAL} ${ITERATIONS}
$ prstat -mvcn 5 10 3
```

```
PID  USERNAME  USR  SYS  TRP  TFL  DFL  LCK  SLP  LAT  VCX  ICX  SCL  SIG  PROCESS/NLWP
26649 root 5.9 17 1.0 12 45 0.0 19 0.1 2K 84 47K 0 prstat/1
26237 root 0.3 0.1 0.0 0.7 1.3 0.0 98 0.0 72 5 493 0 sshd/1...
```

The first iteration of prstat includes CPU data from before the start of prstat. In general, for "current" processor usage, review the second and subsequent iterations.

Be careful of relying upon any interpretation of prstat without it operating in -m 'micro-stat' mode, since there is no accurate timebase to the intervals against which percentage calculations can ever be accurately maintained.

**Per-thread CPU usage**

Use the -L flag along with -p $PID to display accumulated CPU time and CPU usage by thread (light-weight process [LWP]):

```
$ prstat -mvcLn ${MAXTHREADS} -p ${PID} ${INTERVAL} ${ITERATIONS}
$ prstat -mvcLn 50 -p 1089 10 12
```

```
PID  USERNAME  SIZE   RSS  STATE   PRI  NICE   TIME   CPU  PROCESS/LWPID
1089 noaccess 119M 100M sleep 59 0 3:12:24 0.0% java/14
1089 noaccess 119M 100M sleep 59 0 1:55:58 0.0% java/35
1089 noaccess 119M 100M sleep 59 0 0:00:00 0.0% java/38
1089 noaccess 119M 100M sleep 59 0 0:00:00 0.0% java/36...
```

prstat -L for threads has similar behavior to prstat for processes. Without -mv, it reports damped average % CPU. With -mv, the first iteration includes CPU data from before the start of prstat.

**CPU Statistics**

Query available CPU statistics:

```
# cpustat -h
...
```

```
event specification syntax:
[picn=]<eventn>[,attr[n][=<val>]][,[picn=]<eventn>[,attr[n][=<val>]],...]
event0: Cycle_cnt Instr_cnt Dispatch0_IC_miss IC_ref DC_rd DC_wr...
```
Query CPU statistics:

```bash
# cpustat -c EC_ref,EC_misses 5 2
```

```
<table>
<thead>
<tr>
<th>time</th>
<th>cpu</th>
<th>event</th>
<th>pic0</th>
<th>pic1</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.011</td>
<td>0</td>
<td>tick</td>
<td>2037798</td>
<td>90010</td>
</tr>
<tr>
<td>5.011</td>
<td>1</td>
<td>tick</td>
<td>1754067</td>
<td>85031</td>
</tr>
<tr>
<td>10.011</td>
<td>1</td>
<td>tick</td>
<td>2367524</td>
<td>101481</td>
</tr>
<tr>
<td>10.011</td>
<td>0</td>
<td>tick</td>
<td>4272952</td>
<td>195616</td>
</tr>
<tr>
<td>10.011</td>
<td>2</td>
<td>total</td>
<td>10432341</td>
<td>472138</td>
</tr>
</tbody>
</table>
```

The cputrack command is basically the same as cpustat but works on a per-process level.

### Interrupts

Interrupt statistics can be queried with intrstat:

```bash
$ intrstat 5 2
```

```
<table>
<thead>
<tr>
<th>device</th>
<th>cpu0 %tim</th>
<th>cpu1 %tim</th>
</tr>
</thead>
<tbody>
<tr>
<td>bge#0</td>
<td>0 0.0</td>
<td>4 0.0</td>
</tr>
<tr>
<td>glm#0</td>
<td>3 0.0</td>
<td>0 0.0</td>
</tr>
<tr>
<td>uata#0</td>
<td>0 0.0</td>
<td>0 0.0</td>
</tr>
</tbody>
</table>
```

```bash
$ vmstat -i
```

```
interrupt     total     rate
-------------------------
clock         3244127300 100
-------------------------
Total         3244127300 100
```

### Hardware Encryption

Recent versions of the IBM SDK that run on Solaris support the hardware encryption capabilities of the Ultra-SPARC T2 CMT processor through the IBMPKCS11Impl security provider which is the first in the java.security provider list:

**Physical Memory (RAM)**

Program memory should not page out of RAM. This can be monitored with the api, apo, and apf columns in `vmstat -p`. For example:

```
# vmstat -p 5 3
memory           page          executable      anonymous      filesystem
swap  free  re  mf  fr  de  sr  epi  epo  epf  api  apo  apf  fpi  fpo
fpf
4902128 1116760 76 851 1 0 0 0 0 0 0 0 0 0 0 1
1
4304784 931536 25 31 0 0 0 0 0 0 0 0 0 0 0 0
0
4304560 931320 447 5117 0 0 0 0 0 0 0 0 0 0 0 2
0
```

The first line of output is a set of statistics from boot and can usually be discarded.


**Input/Output (I/O)**

Query disk usage:

```
$ df -h
Filesystem             size   used  avail capacity Mounted on
/dev/dsk/c1t0d0s0       63G    60G   3.3G    95%    /dev/dsk/c1t0d0s0
/devices                 0K     0K     0K     0%    /devices
ctfs                     0K     0K     0K     0%    /system/contract
proc                      0K     0K     0K     0%    /proc
mnttab                    0K     0K     0K     0%    /etc/mnttab
swap                    4.4G   1.6M   4.4G     1%    /etc/svc/volatile
fd                       0K     0K     0K     0%    /dev/fd
swap                    4.5G   49M   4.4G     2%    /tmp
swap                    4.4G   56K   4.4G     1%    /var/run...
```

When encountering "too many open files" ulimit issues use:

```
lsof -p <pid>
```

Use iostat for basic disk monitoring. For example:

```
$ iostat -xtcn 5 2
tty     cpu
  tin tout  us  sy  wt  id
  0  1  2  2  0  96

  extended device statistics
  r/s  w/s  kr/s  kw/s  wait  actv  wsvc_t  asvc_t  %w  %b  device
  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  1.1  0  0  c0t0d0
  0.5  2.8  4.8  8.6  0.0  0.1  0.0  18.6  0  1  c1t0d0
  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0  0  c1t1d0
  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  4.6  0  0  wassun1:vold(pid463)
```

```
tty     cpu
  tin tout  us  sy  wt  id
  0  98  0  0  0  99
```
extended device statistics

<table>
<thead>
<tr>
<th>r/s</th>
<th>w/s</th>
<th>kr/s</th>
<th>kw/s</th>
<th>wait</th>
<th>actv</th>
<th>wsfc_t</th>
<th>asvc_t</th>
<th>%w</th>
<th>%b</th>
<th>device</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>c0t0d0</td>
</tr>
<tr>
<td>0.0</td>
<td>2.4</td>
<td>0.0</td>
<td>7.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>19.3</td>
<td>0</td>
<td>1</td>
<td>clt0d0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td>clt1d0</td>
</tr>
<tr>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

wassun1:vold(pid463)...

An alternative is fsstat:

```
$ fsstat -F
new  name   name  attr  attr lookup rddir  read  read  write  write
file  remov  get   set   ops   ops   ops   bytes  bytes
7.11M  3.03M  632K  45.6G  35.9G  90.5M  3.97T  906M  241G
ufs   0  0  0  1.48G  0  1.43G  13.0M  723M  254G  46.9K  8.20M
proc  0  0  0  255  0  25  22  0  0  0  0
nfs   0  0  0  0  0  0  0  0  0  0  0
zfs   0  0  0  785M  0  0  0  0  0  0
lofs  239M  13.5M  225M  272M  105K  549M  23.9K  209K  362M  226M  91.6G
tmpfs
0  0  0  10.3M  0  0  0  30  4.27K  0  0
mntfs 0  0  0  0  0  0  0  0  0  0  0
nfs3  0  0  0  0  0  0  0  0  0  0  0
nfs4  0  0  0  0  0  0  0  0  0  0  0
autofs
```

Query swap usage:

```
$ swap -s
total: 876400k bytes allocated + 45488k reserved = 921888k used, 4645872k
available
```

Zettabyte File System (ZFS)

Consider isolating the ZFS intent log to a separate disk.

Networking

Query socket information:

```
$ netstat -an
TCP: IPv4
Local Address                  Remote Address          Swind Send-Q Rwind Recv-Q State
----------------------------- ------------------------ -------- --------- ---------  
* .32772                      * .*                     0        0        49152    0 LISTEN
127.0.0.1.32833               127.0.0.1.32794          32768    0        32768    0 ESTABLISHED...
```

When running into "too many open files" use

```
netstat -an | grep ESTA | wc -l
```

Query socket statistics periodically:

```
$ netstat -i 5 2
input  bge0     output     input (Total)  output
```
Starting with Solaris 11, use dlstat for network utilization (http://docs.oracle.com/cd/E23824_01/html/821-1458/ggjew.html):

```bash
# dlstat -r -i 1
```
```
LINK   IPKTS  RBYTES   INTRS  POLLS  CH<10  CH10-50  CH>50
e1000g0 101.91K  32.86M  87.56K  14.35K  3.70K  205       5
nxge1   9.61M  14.47G  5.79M  3.82M  379.98K  85.66K  1.64K
vnic1       8       336       0       0       0       0       0
```
```
# dlstat -t -i 5
```
```
LINK   OPKTS  OBYTES  BLKCNT UBLKCNT
```
```
e1000g0  40.24K  4.37M       0       0
nxge1   9.76M  644.14M       0       0
vnic1       0       0       0       0
```
```
e1000g0       0       0       0       0
nxge1  26.82K   1.77M       0       0
vnic1       0       0       0       0
```

Query detailed socket statistics:

```bash
# netstat -s
```
```
TCP    tcpRtoAlgorithm     =     4    tcpRtoMin           =   400
tcpRtoMax           = 60000    tcpMaxConn          =    -1
tcpActiveOpens      = 2162575    tcpPassiveOpens     = 349052
tcpAttemptFails     = 1853162    tcpEstabResets      = 19061...
```

Ping a remote host. In general, and particularly for LANs, ping times should be less than a few hundred milliseconds with little standard deviation.

```bash
$ ping -ns 10.20.30.1
PING 10.20.30.1 : 56 data bytes
64 bytes from 10.20.30.1: icmp_seq=0. time=77.9 ms
64 bytes from 10.20.30.1: icmp_seq=1. time=77.2 ms
64 bytes from 10.20.30.1: icmp_seq=2. time=78.3 ms
64 bytes from 10.20.30.1: icmp_seq=3. time=76.9 ms
```

**snoop**


Capture all traffic:

```bash
$ su
# nohup snoop -r -o capture`hostname`_`date +"%Y%m%d%H%M"`.snoop -q -d $
{INTERFACE} &
# sleep 1 && cat nohup.out # verify no errors in nohup.out
```

Page 170
Use Wireshark to analyze the network packets gathered (covered in the Major Tools chapter).

Use -s to only capture part of the packet.

snoop does not have built-in support for log rollover.

**Kernel**

List available kernel statistics:

```
# kstat -l
bge:0:bge0:brdcstrcv
bge:0:bge0:brdcstxmt...
```

Query kernel statistics:

```
# kstat -p -m cpu_stat -s 'intr*'
cpu_stat:0:cpu_stat0:intr    1118170526
cpu_stat:0:cpu_stat0:intrblk    122410
cpu_stat:0:cpu_stat0:intrthread    828519759
cpu_stat:1:cpu_stat1:intr    823341771
cpu_stat:1:cpu_stat1:intrblk    1671216
cpu_stat:1:cpu_stat1:intrthread    1696737858
```

**KSSL**

On older versions of Solaris and older programs linked with older libraries, you may need to enable the KSSL kernel module, if available, to fully utilize hardware encryption (e.g. TLS performance): [http://docs.oracle.com/cd/E19253-01/816-5166/6mbb1kq5t/index.html](http://docs.oracle.com/cd/E19253-01/816-5166/6mbb1kq5t/index.html)

**truss**

Truss can be used to attach to a process and print which kernel/system calls are being made:

```
# truss -p ${PID}
```

Warning: truss can have a large performance effect when used without filters.

**Modifying Kernel Parameters**

Some kernel parameters can be set by modifying the /etc/system file and rebooting ([http://docs.oracle.com/cd/E23824_01/html/821-1450/chapter1-9.html](http://docs.oracle.com/cd/E23824_01/html/821-1450/chapter1-9.html)). For example:

```
set lim_fd_max = 10000
```

Some networking parameters can be set using the ipadm set-prop command. These updates are persisted on reboot (unless the -t option is specified). For example:

```
# ipadm set-prop -p _time_wait_interval=15000 tcp
```

ipadm command: [http://docs.oracle.com/cd/E26502_01/html/E29031/ipadm-1m.html](http://docs.oracle.com/cd/E26502_01/html/E29031/ipadm-1m.html)

The ipadm command replaces the "ndd" command in recent versions of Solaris: [http://docs.oracle.com/cd/E26502_01/html/E28987/gmafe.html](http://docs.oracle.com/cd/E26502_01/html/E28987/gmafe.html)
Note that Solaris 11 changed the names of some of the network tunable parameters:

Networking

Update the TIME_WAIT timeout to 15 seconds by running 
`# ipadm set-prop -p _time_wait_interval=15000 tcp`

Update the FIN_WAIT_2 timeout to 67.5 seconds by running 
`# ipadm set-prop -p tcp_fin_wait_2_flush_interval=67500 tcp`

Update the TCP keepalive interval to 15 seconds by running 
`# ipadm set-prop -p _keepalive_interval=15000 tcp`

Update the TCP listen backlog to 511 by running 
`# ipadm set-prop -p _conn_req_max_q=511 tcp`

Update the maximum send and receive buffer sizes to 4MB by running 
`# ipadm set-prop -p max_buf=4194304 tcp`
(http://docs.oracle.com/cd/E23824_01/html/821-1450/chapter4-31.html)

Update the maximum value of the TCP congestion window to 2MB by running 
`# ipadm set-prop -p _cwnd_max=2097152 tcp`
(http://docs.oracle.com/cd/E23824_01/html/821-1450/chapter4-31.html)

Update the default send window size to 1MB by running 
`# ipadm set-prop -p send_buf=1048576 tcp`
(http://docs.oracle.com/cd/E23824_01/html/821-1450/chapter4-31.html)

Update the default receive window size to 1MB by running 
`# ipadm set-prop -p recv_buf=1048576 tcp`
(http://docs.oracle.com/cd/E23824_01/html/821-1450/chapter4-31.html)

Process Limits

Update the maximum file descriptors to 10,000 by updating these lines in /etc/system and rebooting

```
set lim_fd_max = 10000
set rlim_fd_cur = 10000
```

dtrace

Dtrace is a very powerful, dynamic tracing tool. For more information, see
http://www.solarisinternals.com/wiki/index.php/DTrace_Topics_Intro

Sample 5-level user stack traces for Java processes:
```
# dtrace -n 'profile-1001 /execname == "java"/ { @[ustack(5)] = count(); }`
```
Print a stack trace any time a function is called:

```bash
# dtrace -n 'syscall::read:entry /execname == "bash"/ { ustack(); }'
```

List probes:

```bash
# dtrace -ln 'proc:::'
```

Useful scripts:

- Sample user and kernel CPU stacks:
- Summarize syscalls:
- Track specific syscall times:

DTrace scripts sometimes refer to time in Hertz. To convert: secs = 1/hertz

**FlameGraphs**

```bash
# git clone https://github.com/brendangregg/FlameGraph
# cd FlameGraph
# dtrace -x ustackframes=100 -n 'profile-99 /arg1/ { @[ustack()] = count(); }' tick-60s { exit(0); }' -o out.stacks
# ./stackcollapse.pl out.stacks > out.folded
# ./flamegraph.pl out.folded > out.svg
```

**Logical Domains, Zones, and Processor Sets/Pinning**

Logical domains, or LDOMs, are a way to virtualize the physical hardware to partition it into multiple guest operating system instances. List domains: ldm list-bindings

Non-global zones, or containers, are a way to virtualize an operating system instance further while sharing the base operating system image and runtime (the parent global zone).

Zones can be used to accomplish processor sets/pinning using resource pools. In some benchmarks, one JVM per zone can be beneficial.

- First, stop the non-global zone
- List zones: zoneadm list -vi
- Enable resource pools: svcadm enable pools
- Create resource pool: poolcfg -dc 'create pool pool1'
- Create processor set: poolcfg -dc 'create pset pset1'
- Set the maximum CPUs in a processor set: poolcfg -dc 'modify pset pset1 (uint pset.max=32)'
- Add virtual CPU to a processor set: poolcfg -dc "transfer to pset pset1 (cpu $X)"
- Associate a resource pool with a processor set: poolcfg -dc 'associate pool pool1 (pset pset1)'
- Set the resource set for a zone: zonecfg -z zone1 set pool=pool1
- Restart the zone: zoneadm -z zone1 boot
• Save to /etc/pooladm.conf: pooladm -s
• Display processor sets: psrset
• Show the processor set a process is associated with (PSET column): ps -e -o pid,pset,comm

**HP-UX**

**HP-UX Recipe**

1. **CPU core(s)** should not be consistently saturated.
2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.

Also review the general topics in the **Operating Systems chapter**.

**General**

Review some of the tuning recommendations in the following documentation pages:


Check the BIOS to ensure highest speed:

1. Power Management -> HP Power Profile -> Maximum Performance
2. Power Management -> HP Power Regulator -> HP Static High Performance Mode
3. Advanced Options -> Advanced Performance Tuning Options -> Memory Speed with 2 DIMMS per channel -> Maximum MHz

Consider installing the following generally useful software:


Query basic system information:

$ uname -a; model; machinfo; sysdef; swlist -l
Central Processing Unit (CPU)

Check if hyperthreading is enabled or disabled using `machinfo` and consider enabling/disabling it, if applicable:

Hyperthreading enabled:

```
$ machinfo
LCPU attribute is enabled...
```

Hyperthreading disabled:

```
$ machinfo
LCPU attribute is disabled...
```


```
$ hpux_performance.sh $PID
```

Use the `top` and `vmstat` commands for basic process monitoring.


The `ptree` command is a useful way to visualize the process tree.

For custom columns in `ps`:

```
UNIX95= ps -ef -o pid,pcpu,pri,pset
```

GlancePlus

GlancePlus (license required) is a very useful tool. To run it for a few minutes, use this hpux_glance.sh script: http://www-01.ibm.com/support/docview.wss?uid=swg21127574&aid=3

Caliper

The caliper tool is a native sampling profiler (http://h20566.www2.hpe.com/hpsc/doc/public/display?sp4ts.oid=4268168&docId=emr_na-c04221975&docLocale=en_US). The simplest report is the flat profile:

```
/opt/caliper/bin/caliper fprof --process=all --attach $PID --duration 60 -o fprof.txt
```

System wide:

```
/opt/caliper/bin/caliper fprof -o fprofsystem.txt --ev all -w -e 30
```

Or

```
/opt/caliper/bin/caliper fprof --scope=kernel --duration=60 -o kernelfprof.txt
```
**HPjmeter**


```
$ /opt/hpjmeter/bin/javaGlanceAdviser.ksh $PID
```

"If you also collected GC information using the -Xverbosegc option, you can append the Glance data to the GC log file and then use HPjmeter to read the combined file."

**jps**

Use the `jps` tool to map Java server names to process IDs. Example:

```
$ /opt/IBM/WebSphere/AppServer/java/bin/jps -m
9326 WSPreLauncher -nosplash -application com.ibm.ws.bootstrap.WSLauncher com.ibm.ws.runtime.WsServer
/opt/IBM/WebSphere/AppServer/profiles/node1/config cell1 node1 nodeagent
7113 WSPreLauncher -nosplash -application com.ibm.ws.bootstrap.WSLauncher com.ibm.ws.runtime.WsServer
/opt/IBM/WebSphere/AppServer/profiles/dmgr1/config cell1 dmgr1 dmgr
6283 WSPreLauncher -nosplash -application com.ibm.ws.bootstrap.WSLauncher com.ibm.ws.runtime.WsServer
/opt/IBM/WebSphere/AppServer/profiles/node1/config cell1 node1 server1
```

Or using caliper (on Itanium systems):

```
$ for i in `ps -elfx | grep java | grep -v grep | awk '{print $4}'`; do echo $i; /opt/caliper/bin/caliper fprof --process=root --attach $i --duration 1 | grep Invocation: ; done;
```

**Physical Memory (RAM)**


**Input/Output (I/O)**

Use the `bdf` command to review disk utilization.

**Networking**


Update the TCP keepalive maximum probes by adding "ndd -set /dev/tcp tcp_keepalives_kill 1" to
/etc/rc.config.d/nddconf and running "ndd -c" 

Use the following command to print socket details: `netstat -anf inet`

Ping a remote host. In general, and particularly for LANs, ping times should be less than a few hundred milliseconds with little standard deviation.

```
$ ping -ns 10.20.30.1
PING 10.20.30.1 : 56 data bytes
64 bytes from 10.20.30.1: icmp_seq=0. time=77.9 ms
64 bytes from 10.20.30.1: icmp_seq=1. time=77.2 ms
64 bytes from 10.20.30.1: icmp_seq=2. time=78.3 ms
64 bytes from 10.20.30.1: icmp_seq=3. time=76.9 ms
```

**nettl**


Start capturing all traffic:

```
# nettl -tn all -e all -f networktrace
```

Stop capturing all traffic:

```
# nettl -tf -e all
```

**Profiling**

The JVM on HP supports dynamically enabling and disabling low-overhead sampling profiling using the kill command:

- **Enable Profiling:**
  ```
  $ kill -USR2 PID
  ```
- **Disable Profiling:**
  ```
  $ kill -USR2 PID
  ```

The profiling will write information on each signal to native_stderr.log. For example:

**first signal**

```
eprof: starting profiling Tue Nov 20 14:05:02 2012
eprof: terminating profiling
eprof: cannot measure profiling intrusion
```

**second signal**

```
eprof: writing profile data to
/opt/IBM/WebSphere/AppServer/profiles/node1/java10760_75806.eprof
eprof: done.
```
Modifying Kernel Parameters

Review the following instructions to modify core HP-UX kernel parameters:

Running `ndd -set` will not maintain the parameters after rebooting. Instead, it is recommended to update the parameters in `/etc/rc.config.d/nddconf` and run "ndd -c" to load the values from this file and the values will also be picked up on reboot.

**tusc**

`tusc` is a system call tracer.

```
$ /usr/local/bin/tusc -f -C -o tusc_counts.txt $PID & sleep 30; kill -INT $!
$ /usr/local/bin/tusc -f -l -D -R -T "" -o tusc.txt $PID & sleep 30; kill -INT $!
```

**Processor Sets**

"The default processor set (0) always exists and may not be destroyed. All processes and processors at system init time start out in the system default processor set."

Therefore, you may want to "reserve" processor set 0 for background processes and non-application server JVMs, and only distribute the JVMs across the other processor sets. You should take into account the core, hyperthread, and L3 layout to avoid sharing processors from pset 0 with the JVM processor sets.

List CPU IDs and which processor set IDs they're bound to:

```
$ /usr/sbin/psrset -p
SPU 0          PSET 0
SPU 1          PSET 0
SPU 2          PSET 0
SPU 3          PSET 0
```

Create a processor set for a CPU ID:

```
$ /usr/sbin/psrset -c 1
```

Bind PID to processor set 1:

```
$ /usr/sbin/psrset -b 1 `cat /opt/IBM/WebSphere/AppServer/profiles/node1/logs/server1/*.pid`
```

Query processor sets for PIDs:

```
$ /usr/sbin/psrset -q `cat /opt/IBM/WebSphere/AppServer/profiles/node1/logs/server1/*.pid` `cat /opt/IBM/WebSphere/AppServer/profiles/node1/logs/server2/*.pid`
PID 28493 PSET 0
PID 25756 PSET 0
```
Automation

To assign processor sets automatically, you will need to modify the Java command line. This means that you will not be able to use the administrative console to start servers (you can still use it to stop servers)

1. For each application server instance, run startServer.sh $NAME -script to generate its start script.
2. Now you should have start_$JVMID.sh script for each JVM.
3. Edit each start_...sh script and you should see an exec java line at the bottom. Update to redirect output:
   exec "/opt/IBM/WebSphere/AppServer/java/bin/java" $DEBUG "-XX:...>
   Changes to:
   exec "/opt/IBM/WebSphere/AppServer/java/bin/java" $DEBUG "-XX:... >>
   /opt/IBM/WebSphere/AppServer/profiles/node1/logs/dynamiccluster1_node1/native_stdout.log 2>>
   /opt/IBM/WebSphere/AppServer/profiles/node1/logs/dynamiccluster1_node1/native_stderr.log &
4. Start the JVM in the processor set with (each Nth JVM will have _N in the shell script name) -- replace 1 with the processor set ID:
   /usr/sbin/psrset -e 1 ./start_server1.sh

... 

macOS

Overview of performance analysis tools:  
https://developer.apple.com/library/content/documentation/Performance/Conceptual/PerformanceOverview/PerformanceTools/PerformanceTools.html#/apple_ref/doc/uid/TP40001410-CH205-SW2

Activity Monitor

Activity Monitor is a graphical tool to look at CPU, Memory, and more:  https://support.apple.com/en-us/HT201464

Memory

Roughly, "available" memory is Free + Inactive + Speculative (if Free has Speculative subtracted as vm_stat does) + File-backed pages

The size of a page on OS X is 4096 bytes.

Wired memory (also called resident memory) stores kernel code and data structures that must never be paged out to disk. Applications, frameworks, and other user-level software cannot allocate wired memory. However, they can affect how much wired memory exists at any time. For example, an application that creates threads and ports implicitly allocates wired memory for the required kernel resources that are associated with them. [...] 

Wired memory pages are not immediately moved back to the free list when they become invalid. Instead they are “garbage collected” when the free-page count falls below the
threshold that triggers page out events. [...]

The active list contains pages that are currently mapped into memory and have been recently accessed.

The inactive list contains pages that are currently resident in physical memory but have not been accessed recently. These pages contain valid data but may be removed from memory at any time.

The free list contains pages of physical memory that are not associated with any address space of VM object. These pages are available for immediate use by any process that needs them.

When the number of pages on the free list falls below a threshold (determined by the size of physical memory), the pager attempts to balance the queues. It does this by pulling pages from the inactive list. If a page has been accessed recently, it is reactivated and placed on the end of the active list. In OS X, if an inactive page contains data that has not been written to the backing store recently, its contents must be paged out to disk before it can be placed on the free list.

https://developer.apple.com/library/content/documentation/Performance/Conceptual/ManagingMemory/Articles/AboutMemory.html

[O]n Mac OS X 10.5 we introduced a new, fifth category of memory, speculative memory, used to hold pages that have been read from disk speculatively.


In Activity Monitor, Cached Files is defined as the following, and experiments show this is approximated by "File-backed pages" in vm_stat:

Cached Files: Memory that was recently used by apps and is now available for use by other apps. For example, if you've been using Mail and then quit Mail, the RAM that Mail was using becomes part of the memory used by cached files, which then becomes available to other apps. If you open Mail again before its cached-files memory is used (overwritten) by another app, Mail opens more quickly because that memory is quickly converted back to app memory without having to load its contents from your startup drive.

https://support.apple.com/en-us/HT201464#memory

Detailed memory statistics:
https://developer.apple.com/library/content/documentation/Performance/Conceptual/ManagingMemory/Articles/VMPages.html#/apple_ref/doc/uid/20001985-CJBJFIDD
1. Tune the maximum Java heap size (-Xmx):
   1. Enable verbose garbage collection (-verbose:gc) which prints statistics on garbage collection to files and generally has an overhead less than 1%. Use a tool such as the IBM Garbage Collection and Memory Visualizer to analyze the verbosegc output. The proportion of time spent in garbage collection versus application processing time should generally be less than 10% and ideally less than 1%.
   2. Garbage collection will adapt heap size to keep occupancy between 40% and 70%. Heap occupancy over 70% causes frequent GC cycles... Heap occupancy below 40% means infrequent GC cycles, but cycles longer than they needs to be... The maximum heap size setting should therefore be 43% larger than the maximum occupancy of the application.
2. Consider the particular type of garbage collector to use (see the comparison table in either the IBM Java or Oracle/HotSpot Java chapters).
3. Ensure there is no memory leak with long running tests.
4. If using a generational collector such as IBM gencon/balanced or the Oracle JVM:
   1. Ensure tests run through full/tenured collections and ensure those pause times are not too long.
   2. Ensure that there is a sawtooth pattern in the heap usage after collection. Otherwise, the heap size may be too small or the nursery too big.
   3. Generally, the sawtooth should drop about 25% of the heap size on full collections.
5. Total pause times over a few seconds should be routinely investigated.
6. Use a profiler such as IBM Java Health Center or Java Mission Control with a particular focus on the profiling and lock contention analysis. Otherwise, use periodic thread dumps to review JVM activity with the IBM Thread and Monitor Dump Analyzer tool.
7. Object allocation failures for objects greater than 5MB should generally be investigated.
8. Take a system dump or HPROF heapdump during peak activity and review it with the IBM Memory Analyzer Tool to see if there are any areas in the heap for optimization.
9. Review the stderr and stdout logs for any errors, warnings, or high volumes of messages (e.g. OutOfMemoryErrors).
10. If running multiple JVMs on the same machine, consider pinning JVMs to sets of processor cores and tuning -Xgcthreads/-XcompilationThreads or -XX:ParallelGCThreads.
11. In general, if memory usage is very flat and consistent, it may be optimal to fix -Xms = -Xmx. For widely varying heap usage, -Xms < -Xmx is generally recommended. You may get the best of both worlds by setting -Xms to the lowest steady state memory usage, -Xmaxf1.0 to eliminate shrinkage, -Xminf to avoid compaction before expansion, and -Xmine to reduce expansions.
12. Request a thread dump and search its output for "deadlock" to ensure that no threads are deadlocked (thus reducing throughput).

Additionally, see the chapter for your particular Java vendor:

- IBM Java Runtime Environment
- Oracle Java Runtime Environment
**General**

A Java Virtual Machine (JVM) implements the Java specification and is provided by a vendor such as IBM, Oracle, or an open source project (e.g. OpenJDK, IcedTea). A Java process is a native operating system process within which Java code is executed (thus why a JVM is called a virtual machine). Java objects created by Java applications reside within a Java heap which is a subset of the native heap of the operating system process. For most cases, Java developers do not have to worry about managing the memory of Java objects. Instead, Java periodically collects unused memory by using a garbage collector. Each JVM has different runtime characteristics such as garbage collection policies and tuning.

A Java Software Development Kit (SDK), also known as a Java Development Kit (JDK), provides tools for building Java programs such as the java compiler (javac), as well as the implementation of core Java classes in packages such as java.*. A Java Runtime Environment (JRE) includes both a JVM and the Java classes needed to run a Java program.

If only using IPv4, set the generic JVM argument -Djava.net.preferIPv4Stack=true

**Garbage Collection**

Garbage collection automatically frees unused objects. All major JVMs are designed to work with a maximum Java heap size (specified with -Xmx). When the Java heap is full (or various sub-heaps), an allocation failure occurs and the garbage collector will kick in to try to find space. There are some key aspects to garbage collections:

1. **Mark**: Determine whether objects are live or unused.
2. **Sweep**: Reclaim unused objects by marking their memory as available on a free list.
3. **Compact**: Reduce fragmentation by rearranging the free list into one area of memory.
4. **Generational collector**: Split the heap into two parts: a nursery for short lived objects and a tenured area for long-lived objects.
5. **Copy collection**: Mark, then copy live objects into a survivor space (and/or tenured space for generational collectors). A compaction in the survivor space is implicit.
6. **Stop-the-world (STW) operation**: The "world" is Java and a STW operation stops all Java activity while some operations are performed.

Best practice: The proportion of time spent in garbage collection versus application time should be less than 10% and ideally less than 1%.

One of the most important tuning parameters is the maximum heap size. There are three broad types of memory when considering the maximum heap size:

1. **Base footprint**: This generally includes the base product (such as WAS, Portal, etc.) as well as metadata such as Classes and ClassLoaders used by your application.
2. **Caches and functional queues**: These include in-memory caches such as object caches and pools, and functional queues comprised of queues that hold HTTP session data, for example, if stateful HTTP requests are being used.
3. **Per thread data**: Each piece of work ultimately executes on one or more threads. Each thread will allocate memory while it processes its unit of work. The maximum thread pool size is intimately related to the maximum heap size.

Increasing the maximum heap size increases the time between allocation failures but also increases the
duration of each garbage collection. These two aspects must be kept in balance.

Generational collectors (e.g. IBM gencon/balanced and all HotSpot collectors) split the heap into one or more regions for different age groups of objects. This is based on the observation that Java programs tend to have two different types of objects: long-lived and short-lived. The purpose of splitting the heap (and collecting the heaps in different phases) is to reduce the average time spent in garbage collection by avoiding checking objects that are long-lived since they are less likely to be garbage.

Some tools will refer to "used" heap. This is not necessarily the same as "live" heap or "footprint." This is because some garbage collection policies such as generational collectors will actively avoid collecting certain subsets of garbage in some types of collections. This garbage will still be part of "used" heap, but it is not live, by definition.

Heuristic: If live occupancy is too low (<50%), GC cycles take longer than necessary. If live occupancy is too high, then time between GCs will be too low.

"Look for peaks in the "Pause times (including exclusive access)" line to identify long garbage collection cycles. When you have identified a long garbage collection cycle, determine which of the mark, sweep, and compact activities of the garbage collection cycle caused the cycle to be as long as it was... If you find long garbage collection cycles you can examine, the raw verbose:gc entry for that garbage collection cycle by selecting the first tab at the bottom of the main panel. This tab has the same name as the file containing the verbose:gc data. You can then look for the garbage collection cycle. Raw verbose:gc cycle output is useful because it often contains the reason why particular actions were taken in that cycle and you can see how to avoid those actions."

"To ensure that the occupancy does not exceed 70%, set the maximum Java heap size to at least 43% larger than the Maximum occupancy value provided by GCMV. This setting then makes the Maximum value 70% of the Java heap and the average to be above 40% of the Java heap size... In situations where memory occupancy of the Java heap varies significantly, you might not be able to maintain occupancy between 40% and 70% of the Java heap. In these situations, it is more important to keep the occupancy below 70% of the maximum heap size than it is to keep the occupancy above 40%.

Heuristic: It is important to monitor the maximum garbage collection pause time (particularly for tenured collections). Generally, pause times greater than 1 second may be a problem.

Heuristic: It is important to monitor the maximum size of the requested object causing the allocation failure. Generally, objects greater than 10MB may be a problem.

"Two additional metrics to key in on are the garbage collection intervals and the average pause times for each collection. The GC interval is the amount of time in between garbage collection cycles. The pause time is the amount of time that a garbage collection cycle took to complete... As heap size increases, the interval between GCs increase, enabling more work to be performed before the JVM pauses to execute its garbage collection routines. However, increasing the heap also means that the garbage collector must process more objects and, in turn, drives the GC pause times higher... The GC intervals and pause times together make up the amount of time that was spent in garbage collection: % Time in GC = (Average Pause Time) / (GC Interval + Average Pause Time)"

One useful set of tests is to plot maximum heap size along with % Time in GC to find the best maximum heap size.

One of the most important factors for choosing a policy is the worst case pause time.
Optimal Heap Size

"If the occupancy of the Java heap is too high, garbage collection occurs frequently. If the occupancy is low, garbage collection is infrequent but lasts longer... Try to keep the memory occupancy of the Java heap between 40% and 70% of the Java heap size... The highest point of occupancy of the Java heap is preferably not above 70% of the maximum heap size, and the average occupancy is between 40% and 70% occupancy. If the occupancy goes over 70%, resize the Java heap."

"A correctly sized Java heap should always have a memory occupancy of between 40% and 70% of the maximum Java heap size. To ensure that the occupancy does not exceed 70%, set the maximum Java heap size to at least 43% larger than the Maximum occupancy value provided by GCMV. This setting then makes the Maximum value 70% of the Java heap and the average to be above 40% of the Java heap size."

By default the JVM provides a very flexible heap configuration that allows the heap to grow and shrink dynamically in response to the needs of the application. This allows the JVM to claim only as much memory as necessary at any given time, thereby cooperating with other processes running on the system. The starting and maximum size of the heap can be specified with the -Xms<size><M|G> and -Xmx<size><M|G> options respectively. This flexibility however comes at a cost, as the JVM must request memory from the operating system whenever the heap needs to be grown and return memory whenever it shrinks. This behavior can lead to various worse-case scenarios. If the application's heap requirements oscillate it may cause excessive heap growth and shrinkage. If the JVM is running on a dedicated machine or memory is otherwise not a concern, the overhead of heap resizing can be eliminated by requesting a constant sized heap. This can be accomplished by setting -Xms equal to -Xmx. Choosing the right size for the heap is very important, as GC overhead is directly proportional to the size of the heap! The heap should be large enough to satisfy the application's maximum memory requirements and also contain some wiggle room. The GC has to work much harder when the heap is near full capacity due to fragmentation and other issues, so 20-30% of extra space above the application's maximum needs can lower overall GC overhead.
If an application requires more flexibility than can be achieved with a constant sized heap it may be beneficial to tune the sizing parameters for a dynamic heap. One of the most expensive GC events is object allocation failure. This occurs when there is not enough contiguous space in the current heap to satisfy the allocation and results in a GC collection and a possible heap expansion. If the current heap size is less than Xmx the heap will be expanded in response to the allocation failure if the amount of free space is below a certain threshold. Therefore, it is important to insure that when an allocation fails the heap is expanded to not only allow the failed allocation to succeed, but also many future allocations, otherwise the next failed allocation could trigger yet another GC collection. This is known as heap thrashing. The -Xminf, -Xmaxf, -Xmine, and -Xmaxe group of options can be used to effect when and how the GC resizes the heap. The -Xminf<factor> option (where factor is a real number between 0 and 1) specifies the minimum free space in the heap; if the total free space falls below this factor the heap is expanded. The -Xmaxf<factor> option specifies the maximum free space; if the total free space rises above this factor the heap is shrunk. These options can be used to minimize heap thrashing and excessive resizing. The -Xmine<size><M|G> and -Xmaxe<size><M|G> options specify the minimum and maximum sizes to shrink and grow the heap by. These options can be used to insure that the heap has enough free contiguous space to allow satisfy a reasonable number of allocations before failure.

In general, if memory usage is very flat and consistent, it may be optimal to fix -Xms=-Xmx. For widely varying heap usage, -Xmx<-Xmx is generally recommended. You may get the best of both worlds by settings -Xms to the lowest steady state memory usage, -Xmaxf1.0 to eliminate shrinkage, -Xminf to avoid compaction before expansion, and -Xmine to reduce expansions.

Regardless of whether or not the heap size is constant, it should never exceed the physical memory available to the process, otherwise the operating system may have to swap data in and out of memory. An application's memory behavior can be determined by using various tools, including verbose GC logs.

"GC will adapt heap size to keep occupancy between 40% and 70%. Heap occupancy over 70% causes frequent GC - reduced performance. Heap occupancy below 40% means infrequent GC cycles, but cycles can be longer than they need to be - longer pause times - Reduced Performance. The maximum heap size should therefore be about 40% larger than the maximum occupancy. Maximum occupancy + 43% means occupancy at 70% of total heap. Example: For 70 MB occupancy, 100 MB Max Heap required, which is 70 MB plus 43% of 70 MB." (wikis/form/api/wiki/6fc1b65d-7b08-41f2-8546-91d353469f22/page/b260d2cf-334a-4af4-82f7-7321349fc08e/attachment/5a0f47fa-a79e-4952-9ae3-c3c57ecbe327/media/Performance%20Tuning%20Fundamentals%20and%20Methodology.pdf)

**Generational Garbage Collectors**

**The Sawtooth**

A generational garbage collector tenures objects from the "young" or "nursery" region of the Java heap into an "old" or "tenured" region of the Java heap. If the rate of garbage creation exceeds the rate at which young/nursery generation scavenges can clear objects before they are tenured, then this garbage builds up in the tenured region. When the tenured region fills up, a full garbage collection is run to clean up this garbage. This pattern may suggest suboptimal tuning; however, it may be unavoidable. This is a very common pattern and produces a "sawtooth" shape of Java heap usage. For example, here is a graph from the Garbage Collection and Memory Visualizer Tool:
In the above graph, there are three different plots:

1. Used heap (after collection) - teal color. This is what most people look at first when analyzing Java heap usage. This is a line plot of the heap usage after any garbage collection, whether nursery or tenured. This shows the classic sawtooth pattern.

2. Used heap (after global collection) - red color. This is a better way to look at the "real" Java heap usage over time. This is a line plot of the heap usage only after full garbage collections. This does not show the build-up of garbage in the tenured area. If the slope of this line is positive, there may be a leak.

3. GC type - green color. The "nursery" line at the top is a solid color and this is expected because nursery scavenges should occur frequently under load. The "global" plot at the bottom shows a few periodic full garbage collections. These will line up with the large drops in heap usage when the build-up of garbage is cleaned up in the tenured area.

The implication of the sawtooth is that it is generally naïve to look at the used heap after any collection or to otherwise sample Java heap usage. In the case of a sawtooth usage pattern, such measurements are likely to include a lot of garbage. This also means that common techniques like "tailing" the verbose garbage collection log must be more sophisticated to look at used heap only after global collections (this may be done with `grep -A ... | grep` for example).

**Verbose garbage collection (-verbose:gc)**

Verbose garbage collection (enabled with the command line option -verbose:gc, known colloquially as verbosegc) prints details about the operations of the garbage collector to a text file. This output can be processed in a tool such as the [IBM Garbage Collection and Memory Visualizer](https://developer.ibm.com/visualizer/) to understand the
proportion of time spent in garbage collection, total pause times, etc. By default, verbosegc is not enabled although IBM recommends it is enabled for most production environments. Each garbage collection cycle prints a handful of lines and most of the data will be calculated regardless during the normal operation of the garbage collector; therefore, the overhead of verbosegc is mostly writing these small bursts of data to disk. The overhead of verbosegc on modern, local disks, is usually less than 1%. One IBM Java verbosegc benchmark with WAS 8.5.5 showed 0.13% overhead. See the verbosegc section of each JVM vendor's chapter for more details on overhead, options such as rollover, etc.

- IBM Java verbosegc
- Oracle Java verbosegc

Enabling Verbosegc

Verbosegc may be enabled at runtime through the Memory MBean Verbose attribute (MBeans > java.lang > Memory > Attributes > Verbose > Value = true; Enter):

![Verbosegc Configuration](image)

GC Threads

The garbage collector used by the JVM takes every opportunity to exploit parallelism on multi-CPU machines. All phases of the GC can be executed in parallel with multiple helper threads dividing up the work in order to complete the task as quickly as possible. Depending on the GC strategy and heap size in use, it may be beneficial to adjust the number of threads that the GC uses. The number of GC threads can be specified with the `-Xgcthreads<number>` option. The default number of GC threads is equal to the number of logical processors on the machine minus 1 and it is usually not helpful to exceed this value, reducing it however will reduce GC overhead and may be desirable in some situations. The most important consideration is the number of CPUs available to the JVM; if the JVM is pinned to less than
the total number of CPUs (for example by using execrset on AIX or taskset on Linux) then the number of GC threads should be adjusted. Tuning the number of GC threads may also be desirable when running multiple JVMs on a single machine, or when the JVM is running in a virtualized environment.

### Memory Leaks

Memory leaks in the Java language are a dangerous contributor to garbage collection bottlenecks. Memory leaks are more damaging than memory overuse, because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until the heap is exhausted and the Java code fails with a fatal out-of-memory exception. Memory leaks occur when an unused object has references that are never freed. Memory leaks most commonly occur in collection classes, such as Hashtable because the table always has a reference to the object, even after real references are deleted.

High workload often causes applications to crash immediately after deployment in the production environment. If an application has memory leaks, a high workload can accelerate the magnification of the leakage and cause memory allocation failures to occur.

The goal of memory leak testing is to magnify numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts between expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easier identification of inconsistencies. The following list provides insight on how to interpret the results of your memory leak testing:

Memory leak problems can manifest only after a period of time, therefore, memory leaks are found easily during long-running tests. Short running tests might provide invalid indications of where the memory leaks are occurring. It is sometimes difficult to know when a memory leak is occurring in the Java language, especially when memory usage has seemingly increased either abruptly or monotonically in a given period of time. The reason it is hard to detect a memory leak is that these kinds of increases can be valid or might be the intention of the developer. You can learn how to differentiate the delayed use of objects from completely unused objects by running applications for a longer period of time. Long-running application testing gives you higher confidence for whether the delayed use of objects is actually occurring.

Repetitive test
In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the gap is multiplied in a very progressive way. This testing helps if the number of leaks caused by the execution of a test case is so minimal that it is hardly noticeable in one run.

You can use repetitive tests at the system level or module level. The advantage with modular testing is better control. When a module is designed to keep the private module without creating external side effects such as memory usage, testing for memory leaks is easier. First, the memory usage before running the module is recorded. Then, a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is recorded and checked for significant changes. Remember, garbage collection must be suggested when recording the actual memory usage by inserting System.gc() in the module where you want garbage collection to occur, or using a profiling tool, to force the event to occur.

Concurrency test

Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to memory leaks because of the added complication in the program logic. Careless programming can lead to kept or not-released references. The incident of memory leaks is often facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

Consider the following points when choosing which test cases to use for memory leak testing:

A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, such as adding a new record, creating an HTTP session, performing a transaction and searching a record.
Look at areas where collections of objects are used. Typically, memory leaks are composed of objects within the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the retrieved object.

Heap consumption that indicates a possible leak during periods when the application server is consistently near 100 percent CPU utilization, but disappears when the workload becomes lighter or near-idle, is an indication of heap fragmentation. Heap fragmentation can occur when the JVM can free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but the JVM does not have the time to compact small free memory areas in the heap to larger contiguous spaces.

Another form of heap fragmentation occurs when objects that are less than 512 bytes are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation until a heap compaction occurs.


Many customers have daily or weekly restarts, often because of uninvestigated leaks. These customers will often believe that this is a "solution" to their problem, and although that may avoid OutOfMemoryErrors, it may still impact garbage collection times.

You should also monitor native memory leaks using operating system tools.

**Determining Leaks with Generational Collectors**

By design, generational collectors may put trash into the tenured region until a full collection occurs; therefore, to determine if there is a leak with a generational collector, review the used Java heap after full garbage collections. If the slope is positive, then there may be a leak. However, there are cases where even this pattern may not be a leak. For example, if a SoftReference cache builds up more quickly than the soft reference threshold to clear them, the used heap after global collection may rise but those SoftReferences will be cleared under memory pressure.

** InetAddress Cache**


**32-bit versus 64-bit**

There is a non-trivial cost for 64-bit over 32-bit due to increased memory requirements (larger
pointers/references), reduced processor data and instruction cache line hits (e.g. L2, L3 caches, TLB cache hits), etc. Even with -Xcompressedrefs, the performance hit for 64-bit may be up to 5-10% and the increase in overall memory usage up to 15% (ftp://ftp.software.ibm.com/software/websphere/appserv/was/WAS_V7_64-bit_performance.pdf, ftp://public.dhe.ibm.com/software/webservers/appserv/WAS_64-bit_FAQ.pdf).

Some scenarios where 64-bit is better:

- Computationally expensive or scientific applications.
- A large cache avoids out of process calls to get data.
- If the application requires a large Java heap that cannot be fit into 32-bit processes otherwise.
- If the application requires more native memory.
- If the 64-bit process gets more registers than a 32-bit process, it may run more quickly. For example, with the IBM Power CPU, 32-bit and 64-bit processes get the same number of registers (https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/W51a7ffcf4dfd_4b40_9d82_446ebc23c550/page/Java+Performance+on+POWER7).

### Synchronization and Lock Contention

"If the method is an instance method, [synchronization] locks the lock associated with the instance for which it was invoked (that is, the object that will be known as this during execution of the body of the method). If the method is static, [synchronization] locks the lock associated with the Class object that represents the class in which the method is defined." (See Java Specification)

"Multithreaded applications apply synchronization (locks) around shared resources to ensure that the state of the resource is consistent and that the state is not changed by one thread while it is read by another thread. When an application is deployed on a larger number of CPUs, and subjected to an increasing load, the demand for shared resources increases. To manage the shared resources, more synchronization locks might be created. These locks can become points of contention, preventing threads from executing at the same time. The result is that the application cannot scale to use all available CPU."

You can reduce the rate of lock contention in two ways:

- Reduce the time that the lock is owned when it is taken; for example, limit the amount of work done under the lock in the synchronized block of code.
- Reduce the scope of the lock; for example, instead of using a single lock for an entire table, use separate locks for each row or column of the table.

A thread must spend as little time as possible holding a lock. The longer a lock is held, the greater the probability that another thread tries to obtain the lock and is forced to wait. Reducing the duration that a lock is held reduces the contention on the lock and allows the application to scale further. If you see a long average hold time for a lock, examine the source code:

- check if code that runs under the lock can be moved outside the lock; for example, the code does not act on the shared resource. In this case, move the code outside the lock to allow it to be run in parallel with other threads.
• check if code that runs under the lock results in a blocking operation; for example, a connection to another process is made. In this case, release the lock before the blocking operation starts.

The locking architecture in an application must be granular enough that the level of lock contention is low. The greater the amount of shared resource that is protected by a single lock, the greater the probability that multiple threads try to access the resource at the same time. Reducing the scope of the resource protected by a lock, and therefore increasing the lock granularity, reduces the level of lock contention and allows the application to scale further.

**ReentrantLock**

The states and owners of java.util.concurrent.locks.ReentrantLock instances are not reported in thread dumps. A system dump or HPROF heapdump can be used with the Memory Analyzer Tool (Open Query Browser > Java Basics > Thread Overview and Stacks) to analyze the exclusiveOwnerThread field of the ReentrantLock to review ownership and contention.

**Deadlocks**

A deadlock occurs when two or more threads are contending on resources in such a way that each thread is preventing the others from continuing. If exactly two threads or processes are contending on resources, the deadlock can be called a "deadly embrace".

In a deadlock, Thread 1 owns the lock on Object A and is trying to acquire the lock on Object B. At the same time, Thread 2 owns the lock on Object B and is trying to acquire the lock on Object A. Neither thread will give up the lock it has, so neither thread can continue. In more complicated forms, the deadlock problem can involve multiple threads and multiple locks. In the case of a Java application, the presence of a deadlock typically leads to most or all of the threads in the application becoming unable to carry out further work as they queue up on the locks involved in the deadlock.

See the Deadlock sections below for each JVM vendor for techniques on determining deadlocks.

**Classloading**

"[Before Java 7], multithreaded custom class loaders could deadlock when they did not have an acyclic delegation model." ([http://docs.oracle.com/javase/7/docs/technotes/guides/lang/cl-mt.html](http://docs.oracle.com/javase/7/docs/technotes/guides/lang/cl-mt.html))

Therefore,

"Currently many class loading interactions are synchronized on the class loader lock." ([http://openjdk.java.net/groups/core-libs/ClassLoaderProposal.html](http://openjdk.java.net/groups/core-libs/ClassLoaderProposal.html))

However,

"The Java SE 7 release includes the concept of a parallel capable class loader." ([http://docs.oracle.com/javase/7/docs/technotes/guides/lang/cl-mt.html](http://docs.oracle.com/javase/7/docs/technotes/guides/lang/cl-mt.html))
But,

WAS currently uses the older synchronized classloader design even in Java 7. In cases where there is significant monitor contention in ClassLoader synchronization, the common root cause of the contention is some repeated pattern of class loads (for example, creating JAXP objects), and it's often possible to cache the results of these loads and avoid the problematic class loads.

**Explicit Garbage Collection (System.gc, Runtime.gc)**

It is generally a malpractice for an application to call System.gc() or Runtime.gc() (hereafter referring to both as System.gc(), since the former simply calls the latter). By default, these calls instruct the JVM to perform a full garbage collection, including tenured spaces and a full compaction. These calls may be unnecessary and may increase the proportion of time spent in garbage collection than otherwise would have occurred if the garbage collector was left alone.

The generic JVM arguments -Xdisableexplicitgc (IBM) and -XX:+DisableExplicitGC (HotSpot) are used to tell the JVM to do nothing when System.gc() and Runtime.gc() are called. These arguments are often recommended when it is found in verbosegc that calls to System.gc are negatively affecting the JVM. However, there are potential unintended consequences: For example, in some JVM implementations, core Java functionality such as DirectByteBuffer cleanup may be affected in some situations, leading to unnecessary OutOfMemoryErrors and crashes since the self-healing calls to System.gc to cleanup iceberg native objects have no effect.

Therefore, it is a malpractice to use -Xdisableexplicitgc or -XX:+DisableExplicitGC permanently. The best practice is to figure out who is calling System.gc and avoid or remove those calls. Here are methods to determine this:

Method #1: IBM Java only: Use -Xtrace trigger

Restart the JVM with the generic JVM argument
- Xtrace:trigger=method{java/lang/Runtime.gc,jstacktrace},print=mt

Any time System.gc is called, a stack trace will be printed to native_stderr.log. For example:

```
12:02:55.436*0x191de00 mt.2 > java/lang/Runtime.gc()V Native method, This = 1b24188
12:02:55.463 0x191de00 mt.18 - Instance method receiver:
java/lang/Runtime@00002B8F6249AA70 arguments: ()
12:02:55.463 0x191de00j9trc_aux.0 - jstacktrace:
12:02:55.464 0x191de00j9trc_aux.1 - [1] java.lang.Runtime.gc (Native Method)
12:02:55.464 0x191de00j9trc_aux.1 - [2] java.lang.System.gc (System.java:270)
12:02:55.464 0x191de00j9trc_aux.1 - [3] Test.main (Test.java:3)
```

Important Note: Until IBM Java 7.1, using -Xtrace:print=mt may have a significant overhead. See the -Xtrace section in the IBM Java chapter.

Method #2: Use a tracing profiler

There are many tracing profilers which can time method calls. Find a profiler with the option of only profiling the Runtime.gc method and with the option of getting a call stack to the profile samples.
Method #3: Attach a debugger

Attach a debugger and set a breakpoint in the Runtime.gc method. Then inspect the call stack.

**Common Callers of System.gc**

If using RMI, a background thread calls System.gc every minute. This interval may be controlled with JVM parameters (in milliseconds):

Every 100 hours:

- `Dsun.rmi.dgc.client.gcInterval=360000000`
- `Dsun.rmi.dgc.server.gcInterval=360000000`

Essentially never:

- `Dsun.rmi.dgc.server.gcInterval=92237036854775807`
- `Dsun.rmi.dgc.client.gcInterval=92237036854775807`

• [http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/sunrmiproperties.html](http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/sunrmiproperties.html)

In some cases, DirectByteBuffer usage will drive calls to System.gc (see the DirectByteBuffers section below).

**java.nio.DirectByteBuffers**

Before Java 7, there was significant native memory waste for each DirectByteBuffer:

"Prior to the JDK 7 release, direct buffers allocated using java.nio.ByteBuffer.allocateDirect(int) were aligned on a page boundary. In JDK 7, the implementation has changed so that direct buffers are no longer page aligned. This should reduce the memory requirements of applications that create lots of small buffers. Applications that previously relied on the undocumented alignment can revert to previous behavior if they are run with the command line option: -XX:+PageAlignDirectMemory." ([http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html](http://www.oracle.com/technetwork/java/javase/jdk7-relnotes-418459.html))

If you find clusters of System.gc calls, this is often observed with memory pressure on DirectByteBuffers. This is controlled with -XX:MaxDirectMemorySize=\{size[gG|mM|k|k}\} ([http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/xxmaxdirectmemorysize.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/xxmaxdirectmemorysize.html)) or -Dsun.nio.MaxDirectMemorySize=\{BYTES\} ([http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/Dsunniomaxdirectmemorysize.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendixes/cmdline/Dsunniomaxdirectmemorysize.html)). By default, this is a soft limit that starts at 64MB and grows in 32MB chunks. However, when setting one of these properties, this creates a hard limit, so it's important to not set it too low. A common setting is 1GB or -Dsun.nio.MaxDirectMemorySize=1073741824 (or -XX:MaxDirectMemorySize=1G)
Reflection Inflation

When using Java reflection, the JVM has two methods of accessing the information on the class being reflected. It can use a JNI accessor, or a Java bytecode accessor. If it uses a Java bytecode accessor, then it needs to have its own Java class and classloader (sun/reflect/GeneratedMethodAccessor<N> class and sun/reflect/DelegatingClassLoader). These classes and classloaders use native memory. The accessor bytecode can also get JIT compiled, which will increase the native memory use even more. If Java reflection is used frequently, this can add up to a significant amount of native memory use. The JVM will use the JNI accessor first, then after some number of accesses on the same class, will change to use the Java bytecode accessor. This is called inflation, when the JVM changes from the JNI accessor to the bytecode accessor. ([http://www-01.ibm.com/support/docview.wss?uid=swg21566549](http://www-01.ibm.com/support/docview.wss?uid=swg21566549))

The option -Dsun.reflect.noInflation=true enables immediate inflation on all method invocations. In general, inflated Java bytecode accessors are faster than native JNI accessors, at the cost of additional native and Java memory usage.

Serviceability

The IBM JVM provides significant serviceability improvements such as:

- Thread dumps in separate files with much more information (but still lightweight)
- Easily showing stack traces of calls that allocate large objects
- Method trace and triggers to help with things such as getting stack traces of who is calling System.gc

IBM Java Runtime Environment


Open source J9 code: [https://github.com/eclipse/openj9](https://github.com/eclipse/openj9)

IBM Java Runtime Environment Recipe

In addition to the overall recipe in the Java chapter,

1. In most cases, the gencon garbage collection policy works best, with the key tuning being the maximum heap size (-Xmx) and maximum nursery size (-Xmn).
2. Test with large pages (-Xlp). These are enabled by default in recent versions, but may require operating system configuration for full enablement.
3. Test with -Xaggressive.
4. Test with -XtlhPrefetch.
5. Consider enabling IBM Java Health Center (-Xhealthcenter) by default so that you can attach to particular processes if they start to have trouble.
6. If using IBM Java >= 7 SR3, the IBM JCE security provider and recent Intel, AMD, or POWER >= 8 CPUs, then AESNI hardware acceleration for AES encryption and decryption can be
exploited with -Dcom.ibm.crypto.provider.doAESInHardware=true. This can reduce TLS overhead by up to 35%.

7. If the static IP address of the node on which Java is running is unlikely to change, use -Dcom.ibm.cacheLocalHost=true to reduce localhost lookup time.

8. Take a javacore thread dump and review the Java arguments (UserArgs) and Environment Variables sections for uncommon or debug options.

9. If physical memory allows, increase the size of the shared class cache (-Xshareclasses).

**General**

The modern IBM JVM implementation is named J9. The older IBM JVM (Java <= 1.4.2) implementation is named Sovereign.

By default, Java will cache non-localhost lookups; however, localhost lookups are not cached in case localhost changes. In some operating systems or configurations, localhost lookups add significant overhead. If the static IP address of the node on which Java is running is unlikely to change, use -Dcom.ibm.cacheLocalHost=true to reduce localhost lookup time (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tunejvm_v61.html).

**Garbage Collection**

-Xgcpolicy:gencon is the default garbage collection policy starting in Java 6.26 (WAS 8) - it is a copy collector in the nursery area and a mark-sweep-compact collector in the tenured area. Previously, the default policy is -Xgcpolicy:optthruput.

In garbage collection, generally the term parallel means running on multiple threads, and concurrent means running at the same time as the application (i.e. not stop-the-world). Thread local heaps (TLH) are used by each thread for very small objects to reduce cross thread contention (global heap lock).

**Comparing Policies**

<table>
<thead>
<tr>
<th>Generational - most GC pauses are short (nursery/scavenge collections)</th>
<th>IBM Java -Xgcpolicy:gencon</th>
<th>IBM Java -Xgcpolicy:optthrput</th>
<th>IBM Java -Xgcpolicy:optavgpause</th>
<th>IBM Java -Xgcpolicy:balanced</th>
<th>IBM Java -Xgcpolicy:metronome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compaction</th>
<th>Sometimes</th>
<th>Sometimes</th>
<th>Sometimes</th>
<th>Partial, full in overload conditions</th>
<th>Never</th>
</tr>
</thead>
</table>

<p>| Large Heaps (&gt;10GB) | Yes, depending on heap utilization | No | No | Yes | Yes |</p>
<table>
<thead>
<tr>
<th>Benefits</th>
<th>Tries to balance application throughput with low pause times</th>
<th>Tries to optimize application throughput</th>
<th>Tries to flatten out average pause times</th>
<th>Tries to deal with large heaps by breaking memory into many regions. May help with NUMA</th>
<th>Tries to have consistently low pause times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Soft Real Time - all GC pauses are very short (unless cpu/heap exhaustion occurs)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Hard Real Time - requires hard real time OS, all GC pauses are very short (unless CPU/heap exhaustion occurs)</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Potential Consequences</td>
<td>- Long global GC pauses with large heaps</td>
<td>- Occasional long compactions</td>
<td>- Longer average pause times</td>
<td>- Reduced throughput</td>
<td>- Reduced throughput</td>
</tr>
<tr>
<td></td>
<td>- Benefits negated by frequent large object allocations if they are long-lived</td>
<td>- Reduced CPU</td>
<td>- Increased CPU</td>
<td>- Increased CPU</td>
<td>- Increased Heap Usage</td>
</tr>
<tr>
<td>Recommended for</td>
<td>General Use (e.g. Web applications, messaging systems)</td>
<td>Batch applications</td>
<td>Consistent pause time requirement</td>
<td>Large heaps (&gt;10GB)</td>
<td>Very low, consistent GC latency</td>
</tr>
</tbody>
</table>

-Xgcppolicy:gencon

The idea of a generational collector is to divide the heap up into different areas, and collect these areas at different rates. New objects are allocated out of one such area, called the nursery (or newspace). Since most objects in this area will become garbage quickly,
collecting it offers the best chance to recover memory. Once an object has survived for a while, it is moved into a different area, called tenure (or oldspace). These objects are less likely to become garbage, so the collector examines them much less frequently...

IBM's gencon policy (-Xgcpolicy:gencon) offers a generational GC ("gen") on top of [-Xgcpolicy:optavgpause]. The tenure space is collected as described above, while the nursery space uses a copying collector. This algorithm works by further subdividing the nursery area into allocate and survivor spaces... New objects are placed in allocate space until its free space has been exhausted. The application is then halted, and any live objects in allocate are copied into survivor. The two spaces then swap roles; that is, survivor becomes allocate, and the application is resumed. If an object has survived for a number of these copies, it is moved into the tenure area instead.


The default maximum nursery size (-Xmn) in Java 5 is 64MB. The default in Java 6 is 25% of -Xmx. The larger the nursery, the greater the time between collects, the less objects are likely to survive; however, the longer a copy can potentially take. In general the advice is to have as large a nursery as you can afford to avoid full collects - but the full collects shouldn't be any worse than the optavgpause case. The use of concurrent collection is still in place, and the presence of the nursery should be that there's less likelihood of compacting being required in the tenured space.

For -Xgcpolicy:gencon, consider tuning the nursery size (-Xmn) to a larger proportion of -Xmx (the default is 25%). For applications with more short-lived objects, a performance improvement can be seen by increasing the nursery size (http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html#sec3a)

The scenarios where gencon generally falls down is where there are large object allocations that get tenured directly, or where the nursery is too small and objects are copied several times before they "die." In an ideal world, no object is copied more than once - after the first copy it either dies or is tenured because it is long lived.
Tenure age: "Tenure age is a measure of the object age at which it should be promoted to the tenure area. This age is dynamically adjusted by the JVM and reaches a maximum value of 14. An object’s age is incremented on each scavenge. A tenure age of x means that an object is promoted to the tenure area after it has survived x flips between survivor and allocate space. The threshold is adaptive and adjusts the tenure age based on the percentage of space used in the new area."

(http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/understanding/mm_gc_generational_tenure.html)

A high tenure age means the JVM is aggressive about leaving objects in the nursery, trying to let them die there, which is generally healthy, since the JVM observes that it is able to collect most garbage in a scavenge.

As the nursery size increases, the maximum copy count and the adaptive tenure age will trend to 1. Once the application is a self optimizing tenure age of 1 at runtime, it may make sense to set tenurage=1 explicitly to make startup faster. That sets the tenure age where it will end up anyway, and ensures we don't do a lot of copying of "infrastructure" objects allocated at startup. Fix the tenure age, e.g.: -Xgc:scvNoAdaptiveTenure,scvTenureAge=1

A healthy used tenured heap (after collection) will show a sawtooth pattern where garbage collects in tenured continuously until a full collection. If the nursery size is too large (or the overall heap size is too small), then an unhealthy pattern in this plot will lack the sawtooth and you will see a low tenure age. This will caused the JVM to constantly run full collections and may increase the rate of compactions. A rough guide is that the size of the sawtooth drop should be about 25% of -Xmx. The tenured area may grow and shrink by specifying -Xmos and -Xmox.

You want the nursery to be large enough that data is at most copied once. Once that occurs the duration of a nursery collect is largely fixed at the copy time of the data, so after that increasing the nursery size increases the time between nursery collects - and therefore drops the GC overhead, and mostly likely the frequency of global collections as well.

If you've got large amounts of available RAM and process address space, the extreme tuning solution is a very large nursery with a tenure age of 1. This works on the theory that transactional data can only be copied once, and anything surviving two collects should be put into the old generation as its non-transactional (ie, at startup). You can fix the tenure age via a command line option.

There's no easy (low overhead) way of finding out what the average flip count is, but the following will give you a histogram on each scavenge collect:


The maximum nursery size should be greater or equal to the maximum, concurrent transaction data for all threads. The average number of times that non-tenured objects are copied should be ~= 1

https://w3-03.ibm.com/tools/cm/iram/artifact/{D9D29FCF-84FA-B2C4-6922-1EEE32C76B3C}/HitchhikersGuide/WAS.hitchhikers_guide.html#2.17.3.3.genconoutline

To force full GCs after each N scavenges, use -Xgc:maxScavengeBeforeGlobal=N
If you would like to tail the verbosegc log, it is generally recommended to look at free memory after global collections only because scavenges do not touch trash in the tenured region. On Linux, for example:

```bash
$ tail -f native_stderr.log | grep -A 1 "gc-end.*global" native_stderr.log
<gc-end id="1748" type="global" contextid="1741" durationms="670.959"
timestamp="2014-07-02T16:28:22.476">
  <mem-info id="1749" free="156456360" total="311361536" percent="50">
```

**Tilt Ratio**

The tilt ratio is (size of new or allocate space)/(size of survivor space). The tilt ratio starts at 50% and is dynamically updated in an attempt to maximize the time between scavenges:


**Concurrent Marking**

If generational garbage collection is desired but the overhead of concurrent marking, with respect to both the overhead of the marking thread and the extra book-keeping required when allocating and manipulating objects, is not desired then concurrent marking may be disabled with the `-Xconcurrentlevel0` option. This option is appropriate for workloads that benefit from gencon's optimizations for object allocation and lifetimes but also require maximum throughput and minimal GC overhead while application threads are running. In general for both the gencon and optavgpause GC policies, concurrent marking can be tuned with the `-Xconcurrentlevel<number>` option which specifies the ratio between the amounts of heap allocated and the amounts of heap marked. The default value is 8. The number of low priority mark threads can be set with the `-Xconcurrentbackground<number>` option. By default 1 thread is used for concurrent marking.


**-Xgcpolicy:optthruput**

"The simplest possible garbage collection technique is to continue allocating until free memory has been exhausted, then stop the application and process the entire heap. While this results in a very efficient garbage collector, it means that the user program must be able to tolerate the pauses introduced by the collector. Workloads that are only concerned about overall throughput might benefit from this strategy."

"For applications that are willing to trade some overall throughput for shorter pauses...
-Xgcpolicy:optavgpause attempts to do as much GC work as possible before stopping the application, leading to shorter pauses... The same mark-sweep-compact collector is used, but much of the mark and sweep phases can be done as the application runs. Based on the program's allocation rate, the system attempts to predict when the next garbage collection will be required. When this threshold approaches, a concurrent GC begins. As application threads allocate objects, they will occasionally be asked to do a small amount of GC work before their allocation is fulfilled. The more allocations a thread does, the more it will be asked to help out. Meanwhile, one or more background GC threads will use idle cycles to get additional work done. Once all the concurrent work is done, or if free memory is exhausted ahead of schedule, the application is halted and the collection is completed. This pause is generally short, unless a compaction is required. Because compaction requires moving and updating live objects, it cannot be done concurrently."

Optimized for applications with responsiveness criteria. It reduces and makes more consistent the time spent inside the stop-the-world operations by carrying out some of the stop-the-world activity while the application is running. This has an additional overhead. Optavgpause is suited for consistent allocation patterns or when very large objects adversely affect gencon.

-Xgcpolicy:balanced

The balanced GC policy (available starting with Java 7) is suitable for arbitrarily large heaps, and includes various techniques to prevent worst-case pause time from growing linearly with total heap size. Balanced is a generational policy, so as with gencon most collections will be of the nursery space, and thus will be quite brief. An incremental compaction function performs a subset of compaction work during each GC pause, to avoid the very large pause time associated with compacting the entire heap in a single operation. Tenured space collections are performed on sub-areas of the tenured heap, and objects are grouped by lifespan within the heap, to make tenured collections more efficient and brief.
The primary goal of the balanced collector is to amortize the cost of global garbage collection across many GC pauses, reducing the effect of whole heap collection times. At the same time, each pause should attempt to perform a self contained collection, returning free memory back to the application for immediate reuse.

To achieve this, the balanced collector uses a dynamic approach to select heap areas to collect in order to maximize the return-on-investment of time and effort. This is similar to the gencon policy approach, but is more flexible as it considers all parts of the heap for collection during each pause, rather than a statically defined new space.


The balanced policy can better utilize NUMA node groupings.

Balanced GC overview:

-Xgcpolicy:metronome

The metronome GC policy (available in the base VM starting with Java 7) invokes the WebSphere RealTime (WRT) collector. WRT performs GC in small increments using a time-bounded algorithm to ensure that any individual GC pause is very brief. This behavior is suitable for applications needing consistent low latency response times, e.g. financial transaction systems. The trade-off for getting very low GC latency is some increase in CPU and heap consumption. Unlike gencon, optthruput, and optavgpause collectors, GC pause time with WRT does not increase linearly with heap size, so WRT is suitable for use with very large heaps.


Should you set minimum heap equal to the maximum heap?

For generational policies, the guidance is that you should fix the nursery size: -Xmns == -Xmnx, and allow the tenured heap to vary: -Xmos != -Xmox. For non generational you only have a tenured heap, so -Xms != -Xmx applies.

The reason being that the ability to expand the heap adds resilience into the system to avoid OutOfMemoryErrors. If you're then worried about the potential cost of expansion/shrinkage that this introduces by causing compactions, then that can be mitigated by adjusting -Xmaxf and -Xminf to make expand/shrink a rare event.

Long Mark Times

Long mark times can occur for the following reasons:
1. Increase in the number of Objects on the Java heap
2. Increase in the Java heap size
3. CPU contention
4. System paging

An increase in the number of objects on the Java heap or an increase in the Java heap size is typical. They are the two major factors contributing to GC duration; more Java objects take longer to mark, and more Java heap space means more time is required to traverse the larger memory space. CPU contention and system paging are caused by system resource contention, which you can determine if the paging and CPU information is available.

**Long Sweep Times**

Long sweep times can occur for the following reasons:

1. Increase in Java heap size
2. CPU contention
3. System paging

An increase in Java heap size is typical because the major factor contributing to the duration of the sweep phase is the size of the Java heap that must be traversed. If sweep times increase significantly, the most likely cause is system resource contention, which you can determine if the paging and CPU information is available.

**Compaction**

When compactions occur, they use most of the garbage collection cycle. The Garbage Collector avoids compaction where possible. However, when compactions must occur, the raw verbose:gc output contains a message explaining why the compaction occurred.

The most common cause of avoidable long GC cycles is Java heap expansion and shrinkage. When the Java heap shrinks in size, a compaction is probably required to allow the shrinkage to occur. When the Java heap expands, a compaction might occur before the expansion, particularly when the Java heap occupancy is growing rapidly.

A correctly sized Java heap aims to keep the Java heap occupancy between 40% and 70% of the maximum heap size, which are the trigger occupancy levels for heap expansion and shrinkage. If the range of occupancy is too great to stay within the recommended range, it is more important to keep the occupancy under 70% of the maximum than it is to stay over 40%.

You can remove or reduce the need for shrinkage by increasing the -Xmaxf option from its default value of 0.6, which controls the 40% threshold for shrinkage. By increasing the -Xmaxf value, the lower threshold is reduced below the normal range of occupancy, and shrinkage can be avoided during the normal operation of the application, while still leaving the possibility of shrinkage if the Java heap occupancy drops dramatically.
The `-Xmaxf` parameter specifies the amount of the Java heap that must be free before shrinkage occurs, so a setting of `-Xmaxf0.7` will cause shrinkage when the occupancy is below 30% (70% is free), and `-Xmaxf0.9` cause shrinkage when the occupancy is below 10% (90% is free).

Explicit requests for garbage collection to run using calls to the `System.gc()` or `Runtime.gc()` methods cause a compaction to occur during the garbage collection cycle if compaction did not occur in the previous garbage collection cycle. Explicit garbage collection calls cause garbage collection to run more frequently than necessary, and are likely to cause a compaction to occur. Remove explicit garbage collection calls where possible.

To disable heap shrinkage: `-Xmaxf1.0`

**Verbose garbage collection (-verbose:gc)**

See the [verbosegc section](#) in the general Java chapter for background.

One benchmark by the IBM Java team running on WAS 8.5.5, IBM Java 7, and Power hardware shows the overhead of verbosegc with local disks is 0.13% ([http://www.slideshare.net/cnbailey/websphere-technical-university-introduction-to-java-diagnostic-tools](http://www.slideshare.net/cnbailey/websphere-technical-university-introduction-to-java-diagnostic-tools)).

By default, output will be sent to stderr (in WAS, `native_stderr.log`):


Specifying `-Xverbosegclog` implicitly enables `-verbose:gc` and allows you to write verbosegc to a particularly named file instead, along with the option of rotating said files after certain numbers of GC events (this works on all platforms including z/OS):

[http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendices/cmdline/xverbosegclog.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendices/cmdline/xverbosegclog.html). If you are concerned about performance, you can use `-Xverbosegclog` to write the data to a RAMdisk (see your operating system's chapter). If the JVM is unable to create the file (e.g. permissions, disk space, etc.), verbosegc will fall back to stderr.

When using `-Xverbosegclog`, generally you'll want to specify non-unique dump tokens along with a set of historical files so that the logs roll over across process instances (in practice, this means *not* using `%pid` or `%Y%m%d.%H%M%S`). For example:

```
-Xverbosegclog:verbosegc.%seq.log,20,50000
```

If you specify X,Y after the log name, output is redirected to $\{X\}$ number of files, each containing $\{Y\}$ GC cycles. You can only roll-over by the number of GC cycles and not by raw file size; however, garbage collection events are in the same magnitude in size, so you should be able to approximate. As a rough starting point, one GC cycle outputs about 2KB. Therefore, if let's say you wanted to rollover at 100MB, you would do:

\[
A=\text{Desired size in MB} \\
B=\text{Average GC cycle size output in bytes} \\
Y=(A*1024*1024)/B
\]

So, with $A=100$ and $B=2048$, $Y$ would be 51,200, and then you would use:
That would create 20 historical files with roughly 100MB each. If you wanted to better approximate Y, then you need to better understand B. For that, you could do a historical analysis of verbosegc and calculate the median and mean sizes, in bytes, of each GC event, and fiddle around with B until you get close to A per file.

For WAS, use SERVER_LOG_ROOT to write it to the same place as other logs. The following will create up to 20 historical files of roughly 100MB each:

```
-Xverbosegclog:${SERVER_LOG_ROOT}/verbosegc.%seq.log,20,50000
```

Stop-the-world Events

A "stop-the-world" garbage collection event is defined as the time between exclusive-start and exclusive-end verbosegc elements. This includes scavenges.

Time spent unloading classes

If you find long total GC pause times and the break down includes long times in "time spent unloading classes" in GCMV, then there are a few options:

1. Investigate which classes and classloaders are being unloaded and review if creating these can be reduced or avoided (for example, see the discussion on reflection inflation):
   -verbose:class -Xgc:verboseExtensions
2. Consider using -Xgc:classUnloadingKickoffThreshold=N
   (http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html#migrating)
3. Consider using -Xgc:maxScavengeBeforeGlobal=N
4. Consider changing to a different -Xgcpolicy
5. Ensure IBM Java APAR IV49664 is applied: http://www-01.ibm.com/support/docview.wss?uid=swg1IV49664
6. If unloading times increase as the number of class(loader)es increases, test with
   -Xjit:disableCHOpts,disableCHTable or more aggressively (if there are no Java agents),
   -Xjit:disableCHOpts,disableCHTable,noRecompile

Example verbosegc tag showing time spent unloading classes:

```
<classunloading classloaders="325178" classes="905" timevmquiescems="0.000" timetakenms="16990.786" />
```

Exclusive Access Time

Before a garbage collection, the GC requests "exclusive access" to the JVM. Normally, this should take almost no time. This time is not included in the "Total Pause Time" statistic in GCMV (instead there is an Exclusive Access Time statistic). If this is taking a long time, then most likely some other JVM
thread is holding exclusive access for that time. You can determine how long these are by looking for:

```xml
<exclusive-start id="1628" timestamp="2014-03-31T16:13:51.448"
    intervalms="16331.866">
    <response-info timems="1499.726" idlems="999.647" threads="2"
        lastid="000000000FC2C600" lastname="Thread-123" />
</exclusive-start>
```

The only real way to investigate these is to take a core dump by using the -Xdump slow event and setting the threshold below the average timems value; for example:

```
-Xdump:system:events=slow,filter=1000ms,range=1..2
```

Load the dump into IDDE, run "!info lock" and search for this section:

- id: 0x2aaab4000ed0 name: VM exclusive access
- owner thread id: 27707 name: Thread-105
- waiting thread id: 26717 name: defaultJavaTimer-thread-1

The current thread should match the owner thread, so then just run "!info thread" and you'll see the stack (top frame should be in a native method).

**Excessive Garbage Collection**

By default, if the JVM detects "excessive time" spent in garbage collection (default 95%), an OutOfMemoryError is thrown:


The 95% threshold can be changed with -Xgc:excessiveGCratio=90 where 90 is an example different percentage.

**Explicit Garbage Collection (System.gc, Runtime.gc)**

In addition to the cases covered in the general Java chapter, the IBM JVM may explicitly call System.gc in certain situations. For example, if the JVM is unable to get native memory for class(loader) metadata, it will call System.gc in case this indirectly cleans up native resources. In fact, in this case, the JVM calls an internal method so a full GC will occur even if -Xdisableexplicitgc is set. If the JVM runs out of native memory but continues to run and continues to try to allocate native class(loader) metadata, this can cause a full GC storm.

**Garbage Collection Threads**

The maximum number of logical CPU cores is read and fixed at JVM startup by querying the operating system. If the number of logical CPUs decreases at runtime, and -Xgcthreads is not specified, then the JVM may decide to use less CPUs during a garbage collection based on how many are available. If the number of logical CPU cores increases more than the amount at JVM startup, the JVM will not use these additional cores for garbage collection.

**Garbage Collection Notes**

A scavenge which is converted into a global collection collection is called a percolate.
An "aggressive" GC is declared if a previous GC was unable to reclaim sufficient resources. It means that the GC will try as much as it can, including compaction, class unloading, softref clearing, etc. An aggressive collect may also be triggered if two explicit GCs happen back-to-back.

**Just in Time (JIT) Compiler**

The JIT compiler samples Java method execution at runtime and compiles the byte code of more frequently invoked methods into optimized native code.

The -XsamplingExpirationTime${SECONDS} option allows you to disable this background process a certain number of seconds after the JVM starts when you think that the most important JITting has been completed:

d_idle.html. A related option helps control sampling frequency when the JVM is idle:

-Xjit:samplingFrequencyInIdleMode=${ms}

The JIT has two caches: code and data. The code cache holds the actual compiled native code for any methods that are JITted and the data cache is metadata for said code (which is relatively much smaller than the code). If the application uses a lot of classes or classloaders, the JIT code cache may fill up and subsequent JITting is reduced. Check the size of the JIT code cache by taking a javacore and search for the code cache segments. Each row is 8MB. You can increase the code cache segments and total code cache size with -Xcodecache and -Xjit:codetotal (http://www-01.ibm.com/support/docview.wss?uid=swg11V26401). You can also use the -Xjit:verbose option and review if there are any failures due to lack of code cache. There is also -Xjit:dataTotal=XKB. You may also make more room in the JIT code and data caches by excluding some methods from being JITted with -Xjit:exclude.

Starting with IBM Java 626, by default, there may be up to 4 JIT Compilation threads. These can be quite intensive, and if there are many Java processes on a machine, if they happen to run at the same time, the processors may become saturated. In the same way that -Xgcthreads must be considered when running multiple JVMs on a machine, -XcompilationThreads can be reduced:


There is an option to increase the size of the JIT profiling buffer (default 1024):

-Xjit:iprofilerBufferSize=${bytes}

The option -Xjit:noServer may be used to reduce the level of inlining and therefore reduce JIT CPU utilization, although the program may run more slowly. The option -Xjit:virtualizationHighDensity may be used to be even more aggressive in reducing JIT CPU utilization (it is a superset of -Xjit:noServer), although the program may run even more slowly.

Another way to reduce the CPU usage of JIT compilation is to increase the size of the shared class cache (-Xscmx) and consequently the likelihood that Ahead-of-time (AOT) compiled methods can be reused. In general, AOT can be as big as disk space and physical memory support.

By default, the JIT will compile methods after a certain number of invocations. This can be changed with -Xjit:count (use 0 to compile immediately, although this is generally not recommended):

dixes/cmdline/xjit.html

You can create a log of JIT events, which should be inherently bounded in size to just a few tens of
MB, using -Xjit:verbose={compile*},vlog=jitvlog.txt. Example output:

```
+ (AOT load) sun/io/ByteToCharUTF8.reset()V @ 00002AAAB4D9B5A8-
00002AAAB4D9B6C4 compThread 0
#CR 000000000050C100  Compile request rqk=8 j9method=000000000053BF38
java/util/Hashtable.rehash()V
#CR 000000000050C100  Compile request rqk=8 j9method=0000000000520268
java/lang/String.hashCode()I
(warm) Compiling java/util/Hashtutable.rehash()V  t=10 rqk=8
```

**Shared Classes (-Xshareclasses)**

Consider creating a unique shared class cache for every "type" of JVM - for example, application servers, node agents, deployment manager, etc. [http://www-01.ibm.com/support/docview.wss?uid=swg21965655](http://www-01.ibm.com/support/docview.wss?uid=swg21965655)

Sharing classes in a cache can improve startup time and reduce memory footprint. Processes, such as application servers, node agents, and deployment managers, can use the share classes option... The default is enabled. ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tunejvm_v61.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tunejvm_v61.html))

In general, the shared class cache is a good thing to enable and primarily has a positive impact on JVM startup time and memory footprint.

The share classes option of the IBM [JVM] lets you share classes in a cache. Sharing classes in a cache can improve startup time and reduce memory footprint. Processes, such as application servers, node agents, and deployment managers, can use the share classes option.

This option is enabled by default in [WAS]. To clear the cache, either call the app_server_root/bin/clearClassCache utility [when all Java processes are stopped].

If you need to disable the share classes option for a process, specify the generic JVM argument -Xshareclasses:none for that process.


List all shared class caches: java -Xshareclasses:listAllCaches
Delete a shared class cache: java -Xshareclasses:name=${NAME},destroy

The AOT compiler can be more aggressive with -Xaot:forceAOT

**-Xquickstart**

"The IBM® JIT compiler is tuned for long-running applications typically used on a server. You can use the -Xquickstart command-line option to improve the performance of short-running applications, especially for applications in which processing is not concentrated into a few methods."
-Xquickstart causes the JIT compiler to use a lower optimization level by default and to compile fewer methods. Performing fewer compilations more quickly can improve application startup time. When the AOT compiler is active (both shared classes and AOT compilation enabled), -Xquickstart causes all methods selected for compilation to be AOT compiled, which improves the startup time of subsequent runs. -Xquickstart might degrade performance if it is used with long-running applications that contain methods using a large amount of processing resource. The implementation of -Xquickstart is subject to change in future releases.


-Xaggressive

Consider testing with -Xaggressive: "Enables performance optimizations and new platform exploitation that are expected to be the default in future releases."


-XtlhPrefetch

Large Object Area

By default, LOA is only used as a last resort for large objects if there is no room in the rest of the heap. By default an object has to be larger than ~16k-64k to be allocated in LOA, and that only happens if there is not enough space in SOA.

-Xrs

The -Xrs flag is used to disable the default signal handler (for things such as javacores with kill -3, etc.); however, using this option may reduce performance by up to 5% due to the way the JIT works and the way Java uses signals when available for performance boosts.

External Delays

Performance problems can sometimes be caused by the poor responsiveness of external resources that your application is attempting to access. These external resources include database, File I/O, other applications, and legacy systems. To see if the problem is caused by external delays:

- Identify that a number of threads are waiting on external resources and what those resources are, by examining the javacore.txt file that has been collected.
- Profile the responsiveness of the resource to see if response times are longer than expected.
- You can use a method trace to profile when the call to the resource returns, or you can profile the resource being accessed.
Java thread information is displayed in the "THREADS subcomponent" section of the Javadump. The stack trace is provided for each thread, which can be used to determine whether there are any threads waiting on external resources. A thread may wait on an external resource either in a wait, read, or receive method. In this example, the threads are in the Object.wait() method because of a call to AS400ThreadedServer.receive(), which is an external resource:

```
3XMTHREADINFO  "WebContainer : 0" (TID:0x0000000001191E00,
sys_thread_t:0x00000000010955C0, state:CW, native ID:0x0000000000004454)
prio=5
4XESTACKTRACE  at java/lang/Object.wait(Native Method)
4XESTACKTRACE  at java/lang/Object.wait(Object.java:199(Compiled Code))
4XESTACKTRACE  at com/ibm/as400/access/AS400ThreadedServer.receive(AS400ThreadedServer.java:281(Compiled Code))

4XESTACKTRACE  at com/ibm/as400/access/AS400ThreadedServer.sendAndReceive(AS400ThreadedServer.java:419(Compiled Code))
4XESTACKTRACE  at com/ibm/as400/access/BaseDataQueueImplRemote.read(BaseDataQueueImplRemote.java:220(Compiled Code))
4XESTACKTRACE  at com/ibm/as400/access/KeyedDataQueue.read(KeyedDataQueue.java:413(Compiled Code))
4XESTACKTRACE  at com/ibm/testapp/vjops/infra/cdapj/trans/CDAPDataQRouter.readByteBuffer(Bytecode PC:36(Compiled Code))
4XESTACKTRACE  at com/ibm/testapp/vjops/infra/cdapj/trans/CDAPDataQRouter.getMessage(Bytecode PC:28(Compiled Code))
4XESTACKTRACE  at com/ibm/testapp/vjops/infra/cdapj/trans/DataQueueMsgTransactor.doCDAPTransaction(Bytecode PC:175(Compiled Code))
...
```

```
3XMTHREADINFO  "WebContainer : 2" (TID:0x0000000001495100,
sys_thread_t:0x000000000135D6B0, state:CW, native ID:0x000000000000445C)
prio=5
4XESTACKTRACE  at java/lang/Object.wait(Native Method)
4XESTACKTRACE  at java/lang/Object.wait(Object.java:199(Compiled Code))
4XESTACKTRACE  at com/ibm/as400/access/AS400ThreadedServer.receive(AS400ThreadedServer.java:281(Compiled Code))
4XESTACKTRACE  at com/ibm/as400/access/AS400ThreadedServer.sendAndReceive(AS400ThreadedServer.java:419(Compiled Code))
```

Page 210
One of the threads is in BoundedBuffer.waitGet_(), which is an internal resource [and thus not an external delay; in this case the thread is waiting for work]. If the Javadump shows threads that are suspected to be blocking on external resources, the next step is to profile the response time of those resources to see if they are taking a long time.

You can profile the amount of time taken by a method that accesses an external resource by using method trace. Method trace can capture trace data for the JVM, the Java Class Libraries (JCL), and Java application code. You do not need to modify your application to use method trace, which is useful if the source code for the methods of interest is not available. The following resources describe how to activate and control method trace:

... For example, you might profile the "AS400ThreadedServer.receive()" method, using the following command-line options:

```bash
-Xtrace:maximal=mt,output=mtrace#.out,10m,10,methods={com/ibm/as400/access/AS400ThreadedServer.receive*}
```
These options create up to ten files called mtrace#.out, where the # symbol is replaced with a sequence number. Each is up to 10 MB in size. When all ten possible files have been created, the trace engine begins to overwrite the first file in the sequence. You can then format the mtrace#.out files as described in the IBM Diagnostic Guide for Java. These files provide microsecond precision timing information for the entry and exit of each call to the AS400ThreadedServer.receive() method. You can use this information to calculate the average response time and determine if responsiveness is a problem.

Lock Contention

A monitor has a "thin" lock that can be tested efficiently, but which does not support blocking, and -- only when necessary -- an "inflated" lock. The inflated lock is typically implemented using OS resources that can support blocking, but also is less efficient because of the additional path length required when making the calls to the operating system. Because thin locks don't support blocking, spinning is often used such that threads will spin for a short period of time in case the lock becomes available soon after they first try to acquire it.

Analysis of typical locking patterns gives us the insight that spinning helps most cases, but for some specific cases it does not. Before running an application, it is impossible to know for which monitors spinning will not be useful. It is possible, however, to observe monitor usage and identify at run time those monitors for which you do not believe spinning will be helpful. You can then reduce or eliminate spinning for those specific monitors.

The JVM shipped with WebSphere Application Serer V8 includes spinning refinements that capture locking history and use this history to adaptively decide which monitors should use spin and which should not. This can free up additional cycles for other threads with work to do and, when CPU resources are fully utilized, improve overall application performance.


Starting in Java 6.0.1, various improvements were made that are expected to improve CPU effeciency. If CPU utilization decreases but application performance decreases, test with -Xthr:secondarySpinForObjectMonitors. If application performance is affected after the application has run for some time or after a period of heavy load, test with -Xthr:noAdaptSpin. If heap usage is reduced but overall application performance decreases, test -Xlockword:mode=all (http://www.ibm.com/support/knowledgcenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/problem_determination/optimizations_pd.html).

In a javacore, you may see most threads in Conditional Wait (CW) states which you would normally expect to show as Runnable instead. This is "by design" starting in IBM JVM 5. If the top of a thread stack is neither in Object.wait, nor Thread.sleep, nor Thread.join, nor a native method, then the JVM will put the thread into CW state in preparation for the javacore and will return it to Runnable after the javacore is finished. This is done by having all of the aforementioned threads wait for exclusive access to the JVM by waiting on the "Thread public flags mutex lock." This is done to get an internally

Consider upgrading to the latest version of Java because there are often performance improvements in lock contention in the JDK (for example, [http://www-01.ibm.com/support/docview.wss?uid=swg1IV67003](http://www-01.ibm.com/support/docview.wss?uid=swg1IV67003)).

**Lock Reservation**

Synchronization and locking are an important part of any multi-threaded application. Shared resources must be adequately protected by monitors to insure correctness, even if some resources are only infrequently shared. If a resource is primarily accessed by a single thread at any given time that thread will frequently be the only thread to acquire the monitor guarding the resource. In such cases the cost of acquiring the monitor can be reduced with the -XlockReservation option. With this option it is assumed that the last thread to acquire the monitor will likely also be the next thread to acquire it. The lock is therefore said to be reserved for that thread, thereby minimizing its cost to acquire and release the monitor. This option is well-suited to workloads using many threads and many shared resources that are infrequently shared in practice.

**Deadlocks**

The Javadump file that should have been collected contains a 'LOCKS' subcomponent. During the generation of the javacore.txt file, a deadlock detector is run, and, if a deadlock is discovered, it is detailed in this section, showing the threads and locks involved in the deadlock:

```
Deadlock detected !!!
---------------------
Thread "DeadLockThread 1" (0x41DAB100)
  is waiting for:
    sys_mon_t:0x00039B98 infl_mon_t: 0x00039BD8:
        java/lang/Integer@004B2290/004B229C:
          which is owned by:
              Thread "DeadLockThread 0" (0x41DAAD00)
              which is waiting for:
                sys_mon_t:0x00039B40 infl_mon_t: 0x00039B80:
                    java/lang/Integer@004B22A0/004B22AC:
                      which is owned by:
                          Thread "DeadLockThread 1" (0x41DAB100)
```

This example was taken from a deadlock test program where two threads DeadLockThread 0 and DeadLockThread 1 unsuccessfully attempted to synchronize (Java keyword) on two java/lang/Integers.

You can see in the example that DeadLockThread 1 has locked the object instance java/lang/Integer@004B2290. The monitor has been created as a result of a Java code fragment looking like synchronize(count0). This monitor has DeadLockThread 1 waiting to get a lock on the same object instance (count0 from the code fragment). Below the
highlighted section is another monitor locked by DeadLockThread 0 that has DeadLockThread 1 waiting.

**Reflection Inflation**

For a discussion of reflection and inflation, see the general [Java chapter](http://www.ibm.com/support/docview.wss?uid=swg21566549). On the IBM JVM, the option -Dsun.reflect.inflationThreshold=0 disables inflation completely.

The sun.reflect.inflationThreshold property tells the JVM what number of times to use the JNI accessor. If it is set to 0, then the JNI accessors are always used. Since the bytecode accessors use more native memory than the JNI ones, if we are seeing a lot of Java reflection, we will want to use the JNI accessors. To do this, we just need to set the inflationThreshold property to zero. ([http://www-01.ibm.com/support/docview.wss?uid=swg21566549](http://www-01.ibm.com/support/docview.wss?uid=swg21566549))

On IBM Java, the default -Dsun.reflect.inflationThreshold=15 means that the JVM will use the JNI accessor for the first 15 accesses, then after that it will change to use the Java bytecode accessor. Using bytecode accessor currently costs 3-4x more than an invocation via JNI accessor for the first invocation, but subsequent invocations have been benchmarked to be over 20x faster than JNI accessor.

**Advanced Encryption Standard New Instructions (AESNI)**


In some benchmarks, SSL/TLS overhead was reduced by up to 35%.

Use -Dcom.ibm.crypto.provider.AESNITrace=true to check if the processor supports the AES-IN instruction set:

**Large Object Allocation Stack Traces**

For a 5MB threshold:

-Xdump:stack:events=allocation,filter=#5m


For a size range (5 to 6 MB): -Xdump:stack:events=allocation,filter=#5m..6m
Compressed References

64-bit processes primarily offer a much larger address space, thereby allowing for larger Java heaps, JIT code caches, and reducing the effects of memory fragmentation in the native heap. Certain platforms also offer additional benefits in 64-bit mode, such as more CPU registers. However, 64-bit processes also must deal with increased overhead. The overhead comes from the increased memory usage and decreased cache utilization. This overhead is present with every single object allocation, as each object must now be referred to with a 64-bit address rather than a 32-bit address. To alleviate this, the -Xcompressedrefs option may be used, and it is enabled by default in certain release on certain operating systems. When enabled, the JVM will use smaller references to objects instead of 64-bit references when possible. Object references are compressed and decompressed as necessary at minimal cost.

In order to determine the compression/decompression overhead for a given heap size on a particular platform, review verbosegc:

```xml
<attribute name="compressedRefsDisplacement" value="0x0" />
<attribute name="compressedRefsShift" value="0x0" />
```

Values of 0 essentially indicate that no work has to be done in order convert between references. Under these circumstances, 64-bit JVMs running with -Xcompressedrefs can reduce the overhead of 64-bit addressing even more and achieve better performance.

-Xcompressedrefs is enabled by default in Java 6.0.1 SR5 and Java 7 SR4 when the size of the heap allows it. -Xnocompressedrefs can be used to explicitly disable it. On z/OS, before Java 7.1, compressed references was disabled by default, but it could be enabled explicitly.

Some benchmarks show a 10-20% relative throughput decrease when disabling compressed references: "Analysis shows that a 64-bit application without CR yields only 80-85% of 32-bit throughput but with CR yields 90-95%. Depending on application requirements, CR can improve performance up to 20% over standard 64-bit." ([ftp://public.dhe.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf](ftp://public.dhe.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf)). You may be able to recover some of this drop by increasing L2/L3 processor cache sizes. Disabling compressed references will also dramatically increase Java heap usage by up to 70%. Additional background: [http://www-01.ibm.com/support/docview.wss?uid=swg21660890](http://www-01.ibm.com/support/docview.wss?uid=swg21660890)

Starting with Java 8 SR2 FP10, the maximum heap size that supports compressed references was increased from 25GB to 57GB: [http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/preface/changes_80/whatsnew_sr2fp10.html](http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/preface/changes_80/whatsnew_sr2fp10.html)

-Xgc:preferredHeapBase

With compressed references enabled, due to the design of Java, native metadata must all be allocated in the virtual memory range 0-4GB. This includes all native objects backing classes, classloaders, threads, and monitors. If there is insufficient space for additional metadata to be allocated, then a native OutOfMemoryError (NOOM) will be thrown. In general, this can happen for two reasons: 1) there is a class, classloader, thread, or monitor leak, and 2) the Java heap is sharing the 0-4GB space. The first cause can be investigated with the javacore.txt file that's produced with the NOOM by searching for large numbers of these objects.

The second cause is due to the default performance optimizations that Java makes. The location of the
Java heap will affect the type of compression operations that must be performed on each Java pointer reference ([http://www-01.ibm.com/support/docview.wss?uid=swg21660890](http://www-01.ibm.com/support/docview.wss?uid=swg21660890)). If the Java heap can fit completely underneath 4GB, then no "compression" needs to occur - the top 32 bits are simply truncated. Otherwise, for different locations of the Java heap, different arithmetic operations need to be performed. Before APAR IV37797 on non-z/OS operating systems, part of the Java heap would sometimes be allocated below 4GB without the benefit that comes with putting all of it under 4GB. APAR IV37797 changes the behavior for larger heaps to allocate them higher in the virtual memory area by default. On z/OS operating systems, the default was not changed because of operating system considerations. On all operating systems, there are cases where the Java heap will be preferred underneath 4GB and squeeze the metadata space, thus causing NOOMs. One option is to reduce metadata demands, and the second option is to specify where the Java heap should start. Usually, it is sufficient to start the Java heap at the 4GB mark: -Xgc:preferredHeapBase=0x10000000

-Xgc:classUnloadingKickoffThreshold

Classloaders and classes tend to be long lived objects, so they will usually be tenured; however, they also retain native memory. If a full collection does not run for some time and if there are virtual or physical memory pressures, then you can induce full collections using -Xgc:classUnloadingKickoffThreshold:

The command line option -Xgc:classUnloadingKickoffThreshold=<number> tells the system to start a concurrent tenure collection be started every time <number> new class loaders have been created. So, for example, specifying -Xgc:classUnloadingKickoffThreshold=100 will start a concurrent tenure collect whenever a nursery collect notices that 100 new class loaders have been created since the last tenure collection. ([http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html](http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.html))

Method Tracing (-Xtrace methods)

Before IBM Java 7.1, using any method trace may have a significant performance overhead, in some cases up to 40%, and up to 70% during JVM startup. This only affects the -Xtrace "method" option (including simple triggers), not tpnid or other options. This overhead has been mostly removed in Java 7.1.

To use -Xtrace as a simple tracing profiler, see the Java Profilers chapter.

The IBM JVM includes a default -Xtrace which is written to the in-memory ring buffer and dumped out into a Snap.trc file in some situations. This can be disabled with -Xtrace:none.

Use -Xtrace triggers to gather diagnostics when specified Java methods are executed. For example, to take a javacore on the first 1500 executions:

- Xtrace:trigger=method{ilog/rules.factory/IlrReflect.*Listener,javadump,,,1500}

For example, here is a trace that tracks Java socket I/O activity:
Example output:

```
17:11:02.473807000 0x12b83f00   IO.18   Entry
 >IO_Connect(descriptor=353, connect_to(AF_INET6: port=7272 flow=0 addr=...)  
17:11:02.473944000 0x12b83f00   IO.20   Exit   <IO_Connect  
- return =0 
17:11:02.474078000 0x12b83f00   IO.32   Entry
 >IO_Send(descriptor=353, msg=4197800128, len=20, flags=0)  
17:11:02.474117000 0x12b83f00   IO.34   Exit   <IO_Send  
- bytes sent=20 
17:11:02.474124000 0x12b83f00   IO.32   Entry
 >IO_Send(descriptor=353, msg=4197800128, len=193, flags=0)  
17:11:02.474145000 0x12b83f00   IO.34   Exit   <IO_Send  
- bytes sent=193 
17:11:02.474149000 0x12b83f00   IO.32   Entry
 >IO_Send(descriptor=353, msg=4197800128, len=1498, flags=0)  
17:11:02.474171000 0x12b83f00   IO.34   Exit   <IO_Send  
- bytes sent=1498 
17:12:20.422571000 0x13090c00   IO.21   Entry
 >IO_Recv(descriptor=311, buffer=4195936448, len=88, flags=0)  
17:12:20.422577000 0x13090c00   IO.23   Exit   <IO_Recv  
- bytes read=88 
17:11:02.474183000 0x12b83f00   IO.43   Entry
 >IO_Dup2(fd1=290, fd2=353)  
17:11:02.474206000 0x12b83f00   IO.44   Exit   <IO_Dup2  
- error=353 
17:11:02.474209000 0x12b83f00   IO.47   Entry
 >IO_Close(descriptor=353)  
17:11:02.474210000 0x12b83f00   IO.49   Exit   <IO_Close  
- return code=0
```

To format an xtrace output file:

```
java com.ibm.jvm.format.TraceFormat xtrace.out
```

Trace history for a specific thread can be retrieved through jdmpview or IDDE: `!snapformat -t <JVMThread address>`

**Object Request Broker (ORB) and Remote Method Invocation (RMI)**

Default ORB configuration is specified in `${java}/jre/lib/ orb.properties`. Key parameters include

- `com.ibm.CORBA.ConnectTimeout`: Socket connect timeout
- `com.ibm.CORBA.MaxOpenConnections`: maximum number of in-use connections that are to be kept in the connection cache table at any one time
- `com.ibm.CORBA.RequestTimeout`: number of seconds to wait before timing out on a Request message
- `com.ibm.CORBA.ThreadPool.MaximumSize`: Maximum size of the ORB thread pool

Note that in WAS, there is an ORB.thread.pool configuration which is normally used; however, if the
ThreadPool properties are specified in orb.properties, then they override the WAS configuration.

You may see ORB reader threads (RT) and writer threads (WT). For example, here is a reader thread:

3XMTHREADINFO
"RT=265:P=941052;O=0:WSTCPTransportConnection[addr=...,port=2940,local=48884]"
J9VMThread:0x000000000E255600, j9thread_t:0x00002AAAC15D5470,
java/lang/Thread:0x0000000004CF4B4F0, state:R, prio=5
3XMTHREADINFO1 (native thread ID:0x7EFD, native priority:0x5, native policy:UNKNOWN)
3XMTHREADINFO2 (native stack address range from:0x00002AAAD7D6A000, to:0x00002AAAD7DAB000, size:0x41000)
3XMTHREADINFO3 (native thread ID:0x7EFD, native priority:0x5, native policy:UNKNOWN)
4XESTACKTRACE at java/net/SocketInputStream.socketRead0(Native Method)
4XESTACKTRACE at java/net/SocketInputStream.read(SocketInputStream.java:140(Compiled Code))
4XESTACKTRACE at com/ibm/rmi/iiop/Connection.readMoreData(Connection.java:1642(Compiled Code))
4XESTACKTRACE at com/ibm/rmi/iiop/Connection.createInputStream(Connection.java:1455(Compiled Code))
4XESTACKTRACE at com/ibm/rmi/iiop/Connection.doReaderWorkOnce(Connection.java:3250(Compiled Code))
4XESTACKTRACE at com/ibm/rmi/transport/ReaderThread.run(ReaderPoolImpl.java:142(Compiled Code))

These will normally be in R (runnable) state, even if they are just waiting for the incoming message.

The number of Reader Threads (RT) are controlled by the number of active socket connections, not by the ORB thread pool size. For every socket.connect/accept call, an RT gets created and an RT gets removed when the socket closes. RT is not bounded by MaxConnectionCacheSize which is a soft limit - the cache can grow beyond the MaxConnectionCacheSize. Once the cache hits the MaxConnectionCacheSize, the ORB will try to remove stale i.e. unused connections.

The ORB thread pool size will be a cap on the maximum number of Writer Threads (WT), as only up to the number of ORB threads can be writing.

Connection Multiplicity
com.ibm.CORBA.ConnectionMultiplicity: The value of the ConnectionMultiplicity defines the number of concurrent TCP connections between the server and client ORBs. By default this value is set to 1, i.e. there will be only one connection between the server and client ORB and all the requests between the client and server ORB will be multiplexed onto the same connection. This could lead to a performance bottleneck in J2EE deployments where there are a large number of concurrent requests between client & server ORB. (http://www-01.ibm.com/support/docview.wss?uid=swg21669697)

For example, -Dcom.ibm.CORBA.ConnectionMultiplicity=N

Fragment Size

The ORB separates messages into fragments to send over the ORB connection. You can configure this fragment size through the com.ibm.CORBA.FragmentSize parameter.
To determine and change the size of the messages that transfer over the ORB and the number of required fragments, perform the following steps:

In the administrative console, enable ORB tracing in the ORB Properties page. Enable ORBRas diagnostic trace ORBRas=all (http://www-01.ibm.com/support/docview.wss?uid=swg21254706). Increase the trace file sizes because tracing can generate a lot of data. Restart the server and run at least one iteration (preferably several) of the case that you are measuring. Look at the traceable file and do a search for Fragment to follow: Yes.

This message indicates that the ORB transmitted a fragment, but it still has at least one remaining fragment to send prior to the entire message arriving. A Fragment to follow: No value indicates that the particular fragment is the last in the entire message. This fragment can also be the first, if the message fits entirely into one fragment.

If you go to the spot where Fragment to follow: Yes is located, you find a block that looks similar to the following example:

Fragment to follow: Yes
Message size: 4988 (0x137C)
--
Request ID: 1411

This example indicates that the amount of data in the fragment is 4988 bytes and the Request ID is 1411. If you search for all occurrences of Request ID: 1411, you can see the number of fragments that are used to send that particular message. If you add all the associated message sizes, you have the total size of the message that is being sent through the ORB.

You can configure the fragment size by setting the com.ibm.CORBA.FragmentSize ORB custom property.


In general, the ideal fragment size is 0, but that is assuming that there is always a single response per connection, which is often not the case with the default value of ConnectionMultiplicity. One major purpose of a non-zero fragment size is so that one large response does not hog a connection that other responses would like to use. However, if ConnectionMultiplicity is tuned to eliminate connection contention, then use -Dcom.ibm.CORBA.FragmentSize=0

**Interceptors**

Interceptors are ORB extensions that can set up the context prior to the ORB runs a request. For example, the context might include transactions or activity sessions to import. If the client creates a transaction, and then flows the transaction context to the server, then the server imports the transaction context onto the server request through the interceptors.
Most clients do not start transactions or activity sessions, so most systems can benefit from removing the interceptors that are not required.

To remove the interceptors, manually edit the server.xml file and remove the interceptor lines that are not needed from the ORB section.

**ORB IBM Data Representation (IDR)**
ORB 7.1 introduced dramatic performance improvements.

**java.nio.DirectByteBuffer**
The option -Dcom.ibm.nio.DirectByteBuffer.AggressiveMemoryManagement=true may be used to enable a more aggressive DirectByteBuffer cleanup algorithm (which may increase the frequency of System.gc's).

**XML and XSLT**
Profile your application using tools such as the IBM Java Health Center or more simply by taking multiple thread dumps. If you observe significant lock contention on an instance of java/lang/Class and/or significant CPU time in com/ibm/xtq/xslt/* classes, then consider testing the older XSLT4J interpreter to see if you have better results:

From Version 6, the XL TXE-J compiler replaces the XSLT4J interpreter as the default XSLT processor.

The XL TXE-J compiler is faster than the XSLT4J interpreter when you are applying the same transformation more than once. If you perform each individual transformation only once, the XL TXE-J compiler is slower than the XSLT4J interpreter because compilation and optimization reduce performance.

For best performance, ensure that you are not recompiling XSLT transformations that can be reused. Use one of the following methods to reuse compiled transformations:

- If your stylesheet does not change at run time, compile the stylesheet as part of your build process and put the compiled classes on your classpath. Use the org.apache.xalan.xslt.cmdline.Compile command to compile the stylesheet and set the [http://www.ibm.com/xmlns/prod/xltxe-j/use-classpath](http://www.ibm.com/xmlns/prod/xltxe-j/use-classpath) transformer factory attribute to true to load the classes from the classpath.
- If your application will use the same stylesheet during multiple runs, set the [http://www.ibm.com/xmlns/prod/xltxe-j/auto-translet](http://www.ibm.com/xmlns/prod/xltxe-j/auto-translet) transformer factory attribute to true to automatically save the compiled stylesheet to disk for reuse. The compiler will use a compiled stylesheet if it is available, and compile the stylesheet if it is not available or is out-of-date. Use the [http://www.ibm.com/xmlns/prod/xltxe-j/destination-directory](http://www.ibm.com/xmlns/prod/xltxe-j/destination-directory) transformer factory attribute to set the directory used to store compiled stylesheets. By default, compiled stylesheets are stored in the same directory as the stylesheet.
• If your application is a long-running application that reuses the same stylesheet, use the transformer factory to compile the stylesheet and create a Templates object. You can use the Templates object to create Transformer objects without recompiling the stylesheet. The Transformer objects can also be reused but are not thread-safe.
• If your application uses each stylesheet just once or a very small number of times, or you are unable to make any of the other changes listed in this step, you might want to continue to use the XSLT4J interpreter by setting the javax.xml.transform.TransformerFactory service provider to org.apache.xalan.processor.TransformerFactoryImpl.


For additional information, see http://www-01.ibm.com/support/docview.wss?uid=swg21639667

Javacore Thread Dump

Review the native stack traces as well for hot stacks because that might point to some more fundamental issue in the operating system (e.g. malloc contention), etc.

Per-thread CPU usage in javacore (Java 7 SR6, Java 626 SR7, and Java 7.1): A new line has been added to the header section for each thread, giving CPU usage information for that thread (as available from the OS):

```
3XMTHREADINFO      "main" J9VMThread:0x0000000022C80100, j9thread_t:0x0000000000D4E5C0, java/lang/Thread:0x0000000022B96250, state:R, prio=5
3XMJAVAVALTHREAD   (java/lang/Thread getId:0x1, isDaemon:false)
3XMTHREADINFO1     (native thread ID:0xE90, native priority:0x5, native policy:UNKNOWN)
3XMCPUTIME         CPU usage total: 0.249601600 secs, user: 0.218401400 secs, system: 0.031200200 secs
3XMHEAPALLOC       Heap bytes allocated since last GC cycle=25368 (0x6318)
```

Starting with Java 8, CPU usage of JVM-attached threads is tracked by thread category (which can be disabled with -XX:-ReduceCPUMonitorOverhead):

http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/preface/changes_80/whatsnew.html. New lines at the end of the THREADS section in javacore provide the accumulated CPU totals in each category, for example:

```
1XMTHDSUMMARY  Threads CPU Usage Summary
NULL            =========================
1XMTHDCATEGORY All JVM attached threads: 134.253955000 secs
1XMTHDCATEGORY |
2XMTHDCATEGORY +--System-JVM: 8.642450000 secs
2XMTHDCATEGORY |
3XMTHDCATEGORY | +--GC: 1.216805000 secs
2XMTHDCATEGORY |
3XMTHDCATEGORY | +--JIT: 6.224438000 secs
1XMTHDCATEGORY |
2XMTHDCATEGORY +--Application: 125.611505000 secs
```
In the header lines for each thread, an additional field at the end of the 3XMCPUTIME line indicates the current CPU usage category of that thread, for example:

```
3XMTHREADINFO   "JIT Compilation Thread-0 Suspended"
JVMThread:0x00000000F01EB00, jthread_t:0x00000000296A7F8
   java/lang/Thread:0x00000000E0029718, state:R, prio=10
3XMJAVATHREAD   (java/lang/Thread getId:0x4, isDaemon:true)
3XMTHREADINFO1  (native thread ID:0xDFC, native priority:0xB, native
   policy:UNKNOWN, vmstate:CW, vm thread flags:0x01000001)
3XMCPUTIME      CPU usage total: 5.912437900 secs, user: 5.865637600
   secs, system: 0.046800300 secs, current category="JIT"
```


**Stack Size (-Xss)**

If using large stack sizes, consider setting -Xssi as well: http://www-01.ibm.com/support/docview.wss?uid=swg21659956

**Large Pages (-Xlp)**

Details of enabling large pages are on each operating system page. To see whether large pages are enabled on a running JVM, compare pageSize and requestedPageSize in verbosegc:

```
<attribute name="pageSize" value="0x1000" />
<attribute name="requestedPageSize" value="0x20000" />
```

**Oracle Java Runtime Environment**

**Oracle Java Runtime Environment Recipe**

In addition to the overall recipe in the Java chapter,

1. In most cases, the -XX:+UseParallelOldGC garbage collection policy works best, with the key tuning being the maximum heap size (-Xmx) and maximum new generation size (-XX:MaxNewSize).
2. Set -XX:+HeapDumpOnOutOfMemoryError.
3. When using ergonomics, consider tuning -XX:MaxGCPauseMillis and -XX:GCTimeRatio.
4. When fine-tuning is required, consider disabling ergonomics (-XX:-AdaptiveSizePolicy) and tune the SurvivorRatio (-XX:SurvivorRatio).

**General**

The latest version of the Oracle Java Runtime Environment includes HotSpot and JRockit technologies. Use -XX:+PrintFlagsFinal to see all the options the JVM actually starts with.
## Garbage Collection
By default, the collector uses N threads for minor collection where N = # of CPU core threads. Control with -XX:ParallelGCThreads=N

### Comparing Policies

<table>
<thead>
<tr>
<th></th>
<th>HotSpot UseParallelOldGC</th>
<th>Hotspot UseConcMarkSweepGC</th>
<th>HotSpot UseG1GC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generational - most GC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>pauses are short (nursery/scavenge collections)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compaction</td>
<td>Always</td>
<td>Never</td>
<td>Partial</td>
</tr>
<tr>
<td>Large Heaps (&gt;10GB)</td>
<td>Maybe</td>
<td>Maybe</td>
<td>Yes</td>
</tr>
<tr>
<td>Soft Real Time - all GC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>pauses are very short (unless cpu/heap exhaustion occurs)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard Real Time - requires hard real time OS, all GC pauses are very short (unless CPU/heap exhaustion occurs)</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Benefits</td>
<td>Tries to balance application throughput with low pause times</td>
<td>Special circumstances</td>
<td>Strategic direction</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Non-compacting - requires strategy to force compactions</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Hard to tune</td>
<td></td>
</tr>
<tr>
<td>Potential Consequences</td>
<td>Not designed for low latency requirements</td>
<td>- Larger memory footprint (~30%)</td>
<td>Hard to tune</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Reduced throughput</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Longest worst-case pause times (when compaction is unavoidable)</td>
<td></td>
</tr>
<tr>
<td>Recommended for</td>
<td>General Use (e.g. Web applications, messaging systems)</td>
<td>Special circumstances, e.g. SIP-based (voice/video) systems</td>
<td>Large heaps (&gt;10GB)</td>
</tr>
</tbody>
</table>
Ergonomics

Prior to the J2SE platform version 5.0 tuning for garbage collection consisted principally of specifying the size of the overall heap and possibly the size of the generations in the heap. Other controls for tuning garbage collection include the size of the survivor spaces in the young generation and the threshold for promotion from the young generation to the old generation. Tuning required of a series of experiments with different values of these parameters and the use of specialized tools or just good judgment to decide when garbage collection was performing well.

http://www.oracle.com/technetwork/java/ergo5-140223.html

The goal of ergonomics is to provide good performance with little or no tuning of command line options.

http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html#ergonomics

The implementation checks (in this order):

1. If the GC pause time is greater than the pause time goal then reduce the generations sizes to better attain the goal.
2. If the pause time goal is being met then consider the application's throughput goal. If the application's throughput goal is not being met, then increase the sizes of the generations to better attain the goal.
3. If both the pause time goal and the throughput goal are being met, then the size of the generations are decreased to reduce footprint.

-XX:MaxGCPauseMillis=nnn

A hint to the virtual machine that pause times of nnn milliseconds or less are desired. The VM will adjust the java heap size and other GC-related parameters in an attempt to keep GC-induced pauses shorter than nnn milliseconds. Note that this may cause the VM to reduce overall throughput, and in some cases the VM will not be able to meet the desired pause time goal.

By default there is no pause time goal. There are definite limitations on how well a pause time goal can be met. The pause time for a GC depends on the amount of live data in the heap. The minor and major collections depend in different ways on the amount of live data. This parameter should be used with caution. A value that is too small will cause the system to spend an excessive amount of time doing garbage collection.

-XX:GCTimeRatio=nnn

A hint to the virtual machine that it's desirable that not more than 1 / (1 + nnn) of the application execution time be spent in the collector.

For example -XX:GCTimeRatio=19 sets a goal of 5% of the total time for GC and
throughput goal of 95%. That is, the application should get 19 times as much time as the collector.

By default the value is 99, meaning the application should get at least 99 times as much time as the collector. That is, the collector should run for not more than 1% of the total time. This was selected as a good choice for server applications. A value that is too high will cause the size of the heap to grow to its maximum.

http://docs.oracle.com/javase/7/docs/technotes/guides/vm/gc-ergonomics.html

If you have set -Xms != -Xmx, and default or reasonable values of -Xminf/-Xmaxf, yet you see unexpected heap expansions or contractions (particularly during nursery collects), then ergonomics is likely the cause.

Ergonomics may be disabled with -XX:-AdaptiveSizePolicy.

**Default Throughput/Parallel Scavenge Collector (ParallelGC)**

The throughput collector that performs parallel scavenge copy collection on the young generation. This type of garbage collection is the default type on multi-processor server class machines.

Two types of tuning for this collector:

Option 1: Use the default throughput/parallel scavenge collector with built-in tuning enabled.

Starting with Version 5, the Sun HotSpot JVM provides some detection of the operating system on which the server is running, and the JVM attempts to set up an appropriate generational garbage collection mode, that is either parallel or serial, depending on the presence of multiple processors, and the size of physical memory. It is expected that all of the hardware, on which the product runs in production and preproduction mode, satisfies the requirements to be considered a server class machine. However, some development hardware might not meet this criteria.

The behavior of the throughput garbage collector, whether tuned automatically or not, remains the same and introduces some significant pauses, that are proportional to the size of the used heap, into execution of the Java application system as it tries to maximize the benefit of generational garbage collection. However, these automatic algorithms cannot determine if your workload well-suits its actions, or whether the system requires or is better suited to a different garbage collection strategy.

Consult these tuning parameters:

-XX:+UseParallelGC
-XX:+UseAdaptiveSizePolicy
-XX:+AggressiveHeap

Option 2: Use the default throughput/parallel scavenge collector, but tune it manually.
Disadvantages of using the built-in algorithm that is established using the -XX: +UseAdaptiveSizePolicy parameter, include limiting what other parameters, such as the -XX:SurvivorRatio parameter, can be configured to do in combination with the built-in algorithm. When you use the built-in algorithm, you give up some control over determining the resource allocations that are used during execution. If the results of using the built-in algorithm are unsatisfactory, it is easier to manually configure the JVM resources, than to try and tune the actions of the algorithm. Manually configuring the JVM resources involves the use of half as many options as it takes to tune the actions of the algorithm.

Consult these tuning parameters:

-XX:NewRatio=2 This is the default for a server that is configured for VM mode
-XX:MaxNewSize= and -XX:NewSize=
-XX:SurvivorRatio=
-XX:+PrintTenuringDistribution
-XX:TargetSurvivorRatio=

Concurrent low-pause mark-sweep collector (CMS)

This collector is a radical departure from the evolution of generational garbage collection that has underpinned the Hotspot architecture, permitting the overlap of application thread processing with a dedicated low-priority, background garbage collection thread. If your application data is incompatible with the behavior of the default throughput collector, then the concurrent mark-sweep (CMS) collector might be a viable strategy, particularly for application systems that are intolerant of invasive pauses. This collector is particularly helpful with the very large heaps that are used with the 64-bit JVM, or applications that have a large set of long-lived data, also referred to as a large tenured generation, and that maintains comparatively good cache utilization, largely preserving pages of the young generation, even while the background thread must scan through all the pages of the entire heap.

To employ the concurrent mark-sweep collector as the principle housekeeping agent, add this option, instead of any other garbage collection modes, to your JVM configuration.

Consult these tuning parameters:

-XX:+UseConcMarkSweepGC
-XX:CMSInitiatingOccupancyFraction=75
-XX:SurvivorRatio=6
Among the difficulties for tuning with CMS, is that the worst case garbage collection times, which is when the CMS cycle aborts, can take several seconds, which is especially costly for a system that uses CMS precisely to avoid long pauses. Consequently, service level agreements might dictate the use of CMS, because the average or median pause times are very, very low, and the tuning must err on the cautious side to ensure that CMS cycles don't abort. CMS succeeds only when its anticipatory trigger ensures that the CMS cycle always starts early enough to ensure sufficient free resources are available before they are demanded. If the CMS collector is unable to finish before the tenured generation fills up, the collection is completed by pausing the application threads, which is known as a full collection. Full collections are a sign that further tuning is required to the CMS collector to make it better suit your application.

Finally, unlike other garbage collection modes with a compaction phase, the use of CMS theoretically raises the risk of fragmentation occurring with the HotSpot. However, in practice this is rarely a problem while the collection recovers a healthy proportion of the heap. In cases when the CMS fails, or aborts a collection, an alternative compacting garbage collection is triggered. Inevitably any other type of garbage collection incurs a significant invasive pause compared to a normal CMS collection.


Since Java 6, the option -XX:+ExplicitGCInvokesConcurrent may be used to force System.gc's to run concurrently instead of as stop-the-world operations (http://docs.oracle.com/javase/6/docs/technotes/guides/vm/cms-6.html).

The stop-the-world phases of the CMS garbage collector include CMS-remark (https://blogs.oracle.com/poonam/entry/understanding_cms_gc_logs), and CMS-initial-mark (https://blogs.oracle.com/jonthecollector/entry/the_unspoken_cms_and_printgcdetails).

CMS has poor contraction capabilities, partly because it can only compact on the back of a failed CMS, full collection. If fragmentation is high, this can cause CMS to fail more often and cause many full GCs.

"The incremental mode of CMS (i-CMS) has been deprecated and will likely be removed in a future release. It is recommended to use G1 or regular CMS instead of i-CMS."

**CMS Compaction**
"CMS (Concurrent Mark Sweep ) garbage collection does not do compaction."
(http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html)
Garbage-First Garbage Collector (G1GC)

The Garbage-First (G1) garbage collector is fully supported in Oracle JDK 7 update 4 and later releases. The G1 collector is a server-style garbage collector, targeted for multiprocessor machines with large memories. It meets garbage collection (GC) pause time goals with high probability, while achieving high throughput. Whole-heap operations, such as global marking, are performed concurrently with the application threads.

http://www.oracle.com/technetwork/java/javase/tech/g1-intro-jsp-135488.html

Verbose garbage collection (-verbose:gc)

See the verbosegc section in the general Java chapter for background.

Verbose garbage collection is written to stdout (e.g. native_stdout.log).

With Java >= 6 update 4, run with

-verbose:gc -XX:+PrintGCDateStamps -XX:+PrintGCDetails

With Java < 6 update 4, run with

-verbose:gc -XX:+PrintGCTimeStamps -XX:+PrintGCDetails

-XX:+PrintHeapAtGC may be used for additional information although it has a significant overhead.

HP-UX adds the -Xverbosegc option in addition to the existing verbose GC options. This data is more details and can be graphed in HPjmeter.

Send verbose:gc output to a particular log file: -Xloggc:output.log

Starting with versions Java 6 Update 34 and Java 7 Update 2, use -Xloggc:verbosegc.log -XX:
+UseGCLogFileRotation -XX:NumberOfGCLogFiles=10 -XX:GCLogFileSize=100M

Example:

2010-04-22T18:12:27.796+0200: 22.317: [GC 59030K->52906K(97244K), 0.0019061 secs]

If the verbosegc includes "[Full GC (System)" then it was caused by a call to System.gc or Runtime.gc.

For WAS, use SERVER_LOG_ROOT to write to the same directory as other rfies:

-Xloggc:${SERVER_LOG_ROOT}/verbosegc.log -XX:+UseGCLogFileRotation
-XX:NumberOfGCLogFiles=5 -XX:GCLogFileSize=20M -XX:+PrintGCDateStamps -XX:
+PrintGCDetails

CompressedOops

On 64-bit, ensure -XX:+UseCompressedOops is enabled: "Compressed oops is supported and enabled by default in Java SE 6u23 and later"
(https://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html)
Oops stands for ordinary object pointer.

Recent versions of Oracle Java supports -Xmx much larger than 32GB with CompressedOops using -XX:ObjectAlignmentInBytes: https://bugs.openjdk.java.net/browse/JDK-8040176

**Detailed Garbage Collection Tuning**

-

-XX:+AggressiveOpts:

  Turns on point performance optimizations that are expected to be on by default in upcoming releases. The changes grouped by this flag are minor changes to JVM runtime compiled code and not distinct performance features (such as BiasedLocking and ParallelOldGC). This is a good flag to try the JVM engineering team's latest performance tweaks for upcoming releases. Note: this option is experimental! The specific optimizations enabled by this option can change from release to release and even build to build. You should reevaluate the effects of this option with prior to deploying a new release of Java.

  [http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.4](http://www.oracle.com/technetwork/java/tuning-139912.html#section4.2.4)

Consider -XX:+UseTLAB which "uses thread-local object allocation blocks. This improves concurrency by reducing contention on the shared heap lock." ([http://docs.oracle.com/cd/E13209_01/wlcp/wlss30/configwlss/jvmgc.html](http://docs.oracle.com/cd/E13209_01/wlcp/wlss30/configwlss/jvmgc.html))

Setting -XX:MaxTenuringThreshold=0 means that all the objects surviving a minor GC are instantly promoted. This can cause more full GCs.

The -XX:+AlwaysPreTouch option may be used to force the entire Java heap into RAM on startup.

**Permanent Region (permgen)**

HotSpot used to have a dedicated region of the address space called the permanent generation to store things such as class meta-data, interned Strings, and class static variables. This region needed to be manually sized. If the region was exhausted, the JVM would throw an OutOfMemoryError with the message "PermGen space." The PermGen space has been removed in Java 8 ([http://openjdk.java.net/projects/jdk8/milestones](http://openjdk.java.net/projects/jdk8/milestones)) and replaced with the Metaspace (unbounded by default but may be capped with -XX:MaxMetaspaceSize).

Hotspot's representation of Java classes (referred to here as class meta-data) is currently stored in a portion of the Java heap referred to as the permanent generation. In addition, interned Strings and class static variables are stored in the permanent generation. The permanent generation is managed by Hotspot and must have enough room for all the class meta-data, interned Strings and class statics used by the Java application. Class metadata and statics are allocated in the permanent generation when a class is loaded and are garbage collected from the permanent generation when the class is unloaded. Interned Strings are also garbage collected when the permanent generation is GC'ed.

The proposed implementation will allocate class meta-data in native memory and move interned Strings and class statics to the Java heap. Hotspot will explicitly allocate and free the native memory for the class meta-data. Allocation of new class meta-data would be limited by the amount of available native memory rather than fixed by the value of
-XX:MaxPermSize, whether the default or specified on the command line.

http://openjdk.java.net/jeps/122

"The -XX:MaxPermSize= and -Xmx (Maximum Java Heap size) parameters respectively configure the maximum size of the permanent region, where the class code and related data are logically presented as part of the old generation region but are kept physically separate, and the maximum size of the main heap where Java objects and their data are stored either in the young or old generation regions. Together the permanent region and the main heap comprise the total Java heap. An allocation failure in either of these regions either represents the inability to accommodate either all the application code or all the application data, both of which are terminal conditions, that can exhaust available storage, and cause an OutOfMemory error."

Heap Expansion and Contraction

Java heap expansion and contraction is generally controlled by -XX:MinHeapFreeRatio/-Xminf and -XX:MaxHeapFreeRatio/-Xmaxf: http://docs.oracle.com/cd/E19683-01/806-7930/vmoptions-chapter/index.html

However, ergonomics may sometimes render these options moot.

Reference Processing

PhantomReferences are handled differently than finalizers. Queued PhantomReferences are processed on the back of every GC cycle.

By default, there is a single "Reference Handler" thread which processes the ReferenceQueue. Use -XX:+ParallelRefProcEnabled to enable multiple threads for parallel reference queue processing. This may be useful for things such as high DirectByteBuffer allocation and free rates.

DirectByteBuffers may be monitored with the BufferPoolMXBean: http://docs.oracle.com/javase/7/docs/api/java/lang/management/BufferPoolMXBean.html

Safepoints

Safepoints are the internal mechanism by which the JVM tries to pause application threads for operations such as stop-the-world garbage collections. The duration of these safepoint pauses may be printed with:

-XX:+PrintGCApplicationConcurrentTime -XX:+PrintGCApplicationStoppedTime

Although these parameters have "GC" in their name, recent versions of the JVM repurposed these flags and monitor all safepoint pauses. See http://bugs.java.com/view_bug.do?bug_id=6782663

Additional information may be printed with:

-XX:+PrintSafePointStatistics
Thread Priorities
On Linux, use -XX:ThreadPriorityPolicy=1 and run the process as root to allow the use of Java thread priorities: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=4813310

String.substring Performance
HotSpot V7 update 6 introduced a significant change to the implementation of java/lang/String, where calls to substring no longer return a "view" into the String, but instead return a copy (of the substring portion):

- List of bugs fixed in 7u6: http://www.oracle.com/technetwork/java/javase/2col/7u6-bugfixes-1733378.html
- Change request discussing the changes: http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6924259
- Developer mailing list discussing the changes: http://mail.openjdk.java.net/pipermail/core-libs-dev/2013-February/014609.html
- Java Lobby article on the subject: https://dzone.com/articles/changes-stringsubstring-java-7
- Java Performance Tuning article on the subject: http://java-performance.info/changes-to-string-java-1.7-0_06/

If profiling shows significant activity in substring or in array copy, then this may be why. In general, the change is believed to be positive because with the old behavior, the original, potentially very large, String cannot be garbage collected until all substrings are garbage collected. However, if applications use substring heavily, then they may need to be re-coded.

Reflection Inflation
For a discussion of reflection and inflation, see the general Java chapter. On the HotSpot JVM, the option -Dsun.reflect.inflationThreshold=0 creates an inflated Java bytecode accessor which is used on the second and every subsequent method invocation.

DTrace Integration
Newer versions of Java have DTrace integration, but one large limitation is Bug 6617153, which causes DTrace to fail to evaluate Java thread stack names, making jstack nearly useless.

Java Profilers
There are two broad categories of profilers: statistical/sampling profilers which sample call stacks, and tracing profilers which record method entry/exit times. In general, sampling profilers are very low overhead and suitable for production (e.g. IBM Health Center is less than 1%), whereas tracing profilers may be up to 50% or more overhead and generally aren't suitable for production. Imagine that sampling profilers are like taking javacores at a high frequency (with less overhead since the profiler is only sampling call stacks). Tracing profilers are more accurate but produce a lot more data and have to hook deeply into the JVM to get their data, causing the additional overhead.
Whether or not your tests are going well, it is important to plan in at least some basic profiling tests, both for a single user (either sampling or tracing profiler) and for a full stress test (sampling profiler).

**Java Profilers Recipe**

1. In most cases, sampling profilers are used first and tracing profilers are only used for fine grained tuning or deep dive analysis.
2. Analyze any methods that use more than 1% of the reported time in themselves.
3. Analyze any methods that use more than 10% of the reported time in themselves and their children.
4. Analyze any locks that have large contention rates, particularly those with long average hold times.

**Statistical/Sampling Profilers**

**IBM Java Health Center**

The IBM Java Health Center tool is covered in depth in the [Major Tools chapter](http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html).

**HotSpot HPROF**

HPROF is a sampling JVMTI profiler that ships with Java (and therefore with WebSphere). Restart the JVM with `-agentlib:hprof=cpu=samples` when the program stops gracefully, the sample counts will be printed to stdout/stderr.

Example output:

```plaintext
CPU SAMPLES BEGIN (total = 126) Fri Oct 22 12:12:14 2004
rank  self  accum  count  trace  method
 1 53.17% 53.17%  67  300027  java.util.zip.ZipFile.getEntry
 2 17.46% 70.63%  22  300135  java.util.zip.ZipFile.getNextEntry
 3  5.56% 76.19%   7  300111  java.lang.ClassLoader.defineClass2
 4  3.97% 80.16%   5  300140  java.io.UnixFileSystem.list
 5  2.38% 82.54%   3  300149  java.lang.Shutdown.halt0
 6  1.59% 84.13%   2  300136  java.util.zip.ZipEntry.initFields
 7  1.59% 85.71%   2  300138  java.lang.String.substring
 8  1.59% 87.30%   2  300026  java.util.zip.ZipFile.open
 9  0.79% 88.10%   1  300118  com.sun.tools.javac.code.Type$ErrorType.<init>
10  0.79% 88.89%   1  300134  java.util.zip.ZipFile.ensureOpen
```

http://docs.oracle.com/javase/7/docs/technotes/samples/hprof.html

**Java Mission Control (Formerly JRockit Mission Control)**

The Oracle JVM includes the Java Mission Control tool ([http://www.oracle.com/technetwork/java/javaseproducts/mission-control/index.html](http://www.oracle.com/technetwork/java/javaseproducts/mission-control/index.html)) starting with Java 7u40 (enabled with `-XX:+UnlockCommercialFeatures -XX:+FlightRecorder and other options). Java Mission Control is a live monitoring and analysis tool, with focus on gathering the data necessary...
with the lowest possible impact on the running system. This is very similar to IBM Java Health Center with a sampling profiler and other monitoring. Mission Control solves the many issues with VisualVM such as the ability to run in headless mode, smarter defaults (VisualVM doesn't include java* packages), and higher granularity with lower overhead. However, Java L3 in Hursley confirmed that IBM's Java distribution of the Oracle JVM does not include a license to use Mission Control:

"I have received the official word that Oracle's Mission Control product is outside of the terms of the Java technology license that remains in effect since Oracle's acquisition of Sun Micro Systems. Consequently, no, there is no grant to the IBM product stack the access to use Mission Control based on our Java technology license."

The JVM will fail to start with the error "Error: Trying to use 'UnlockCommercialFeatures', but commercial features are not available in this VM."

Highlights:

- Very low overhead
- Suitable for production environments
- More accurate data

It provides the following tools:

- JMX Console
  - Real time monitoring of Java processes.
  - Captures: garbage collection pauses, memory and CPU usage, JMX Beans deployed into JDK MBean server, etc.
- Java Flight Recorder
  - On-demand, records the JVM and application running.
  - Can be run in production systems.
  - Collects: Execution profile, gc statistics, optimization decisions, object allocation, heap statistics, latency events for locks and I/O

**Installing Mission Control**

- It's shipped in Java 7 update 40 and forward, bin/jmc
- It can also be installed over Eclipse 3.8.2/4.2.2/4.3 using the following update site: [http://download.oracle.com/technology/products/missioncontrol/updatesites/base/5.3.0/eclipse/](http://download.oracle.com/technology/products/missioncontrol/updatesites/base/5.3.0/eclipse/)

  Eclipse 4.4 (Luna) unsupported in JMC version 5.3.

**Installing experimental plugins**

There are some experimental plugins that can be evaluated, they are unsupported and not fully tested, so usage would be at own risk.

- Installing into JMC client:
  - Go to Help > Install New Software... and select those you want to try out.
- Installing into Eclipse
  - Use the following update site: [http://download.oracle.com/technology/products/missioncontrol/updatesites/experi](http://download.oracle.com/technology/products/missioncontrol/updatesites/experi)
mental/5.3.0/eclipse/

Note: For getting the current Eclipse update site for installing JMC or the experimental plugins, refer to the main page [http://www.oracle.com/technetwork/java/javaseproducts/mission-control/index.html](http://www.oracle.com/technetwork/java/javaseproducts/mission-control/index.html)

**JMC Versions:**

- Version 5.3 and later works for Hotspot version 7 update 40 and forward.
- If working with JRockit VM, use Mission Control version 4.1

**Supported platforms:**

This is a simplified list of the supported platforms:

<table>
<thead>
<tr>
<th>OS</th>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Linux</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Mac OS X (64 bit only)</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Solaris</td>
<td>N</td>
<td>Y</td>
</tr>
</tbody>
</table>

Look for specific versions at:

- **Client:**
  - [http://www.oracle.com/technetwork/java/javase/overview/jmc-5-3-certified-system-config-2159573.html](http://www.oracle.com/technetwork/java/javase/overview/jmc-5-3-certified-system-config-2159573.html)

- **Server** (the JVM we want to monitor):
  - Java 7: [http://www.oracle.com/technetwork/java/javase/config-417990.html](http://www.oracle.com/technetwork/java/javase/config-417990.html)

  Highlights: Java SE Update 4 and forward can be monitored by Java Mission Control 5.3, Java SE update 3 and below have limited functionality. Versions previous to 7 cannot be monitored by Java Mission Control 5.3

**Starting the client:**

- Double click on the launcher (bin/jmc)
- Or start from command line.
  - Tip: can be started in debug mode with:

    ```
    $ jmc -consoleLog -debug 2>&1
    ```


  - To increase -Xmx of jmc:

    ```
    $ jmc -vmargs -Xmx10g
    ```

**Starting the server (monitored JVM):**

- Add the following parameters to enable Flight Recorder
  - `-XX:+UnlockCommercialFeatures` `-XX:+FlightRecorder`
• No need to add anything if just using the JMX Console.

Starting the JMX Console
• Start the JMC client.
• All JVMs running that can be monitored by Mission Control are listed in the JVM Browser, select the one you want to monitor.

• Right click, select Start JMX Console

Starting a Recording
• From command line:
  • Use the option `-XX:StartFlightRecording` with the appropriate parameters to establish duration, file name for the recording, etc.
  Example:
    `-XX:+UnlockCommercialFeatures -XX:+FlightRecorder -XX:StartFlightRecording=name=jfr,filename=recording.jfr,maxage=1440,settings=profile`
  • More info on the parameters for the `-XX:StartFlightRecording` at http://docs.oracle.com/javase/7/docs/technotes/tools/windows/java.html
• Using Diagnostic Commands
• You can use the `jcmd` tool to control recordings. Reference: [http://docs.oracle.com/javacomponents/jmc-5-3/jfr-runtime-guide/comline.htm#BABBGJCF](http://docs.oracle.com/javacomponents/jmc-5-3/jfr-runtime-guide/comline.htm#BABBGJCF)
• Using the JMC client.
  • Make sure you enabled Flight Recorder (see "Starting the server")
  • Start the client
  • All enabled JVMs running are listed in the JVM Browser, select the one you want to create a flight recording of.

![Oracle Java Mission Control](image1)

• Right click, select Start Flight Recording

![Oracle Java Mission Control](image2)

• Select the appropriate parameters in the Start Flight Recording dialog and click Finish
To stop the recording (if chosen Continuous Recording or if the Recording time specified turned out to be more than needed):
  - Locate the recording in the JVM browser, right click it and select Stop:
Starting a Recording automatically.

- You can create a recording when a JVM exits using the following option:
  - `XX:FlightRecorderOptions=defaultrecording=true,dumponexit=true,dumponexitpath=path`

- Or you can configure triggers on the JMC console through the MBeans tab group, Triggers tab, to start, stop and dump a recording.

The JMX Console

The JMX console is very comprehensive and customizable, and you can easily reset to the defaults by clicking this button:

JMX Console Overview tab
The Overview tab gives you by default the widgets shown above. You can either:

- Remove any of the widgets. Clicking the \( \times \) button on the right side of each one.
- Customize them. By clicking the \( \square \) button on any of the charts to add an attribute or the Add Dial button (green plus sign) on the Dashboard widget.
- Add new charts. Click the Add Chart button to create additional charts.

**MBeans tab group**

Contains two tabs, MBean Browser and Triggers. The MBean Browser tab gives access to available MBeans information and operations. The Triggers tab allows you to configure certain actions automatically under certain conditions. For example, the screenshot below shows a configuration where, an HPROF will be generated when the JVM Process is "Too Low":

![MBeans tab group screenshot](image)
And the "Too Low" criteria is configured on the Rule Details / Condition Tab:
HotSpot VisualVM

VisualVM (http://docs.oracle.com/javase/7/docs/technotes/guides/visualvm/) is shipped with Java (${JAVA}/bin/jvisualvm) and therefore with WebSphere. It provides both a sampling and tracing profiler. VisualVM does not have a headless mode, so you must run the GUI client on the same machine as the target JVM (http://docs.oracle.com/javase/7/docs/technotes/tools/share/jstatd.html). jvisualvm through jstatd does not support remote profiling: "Java VisualVM... cannot profile remote applications." (http://docs.oracle.com/javase/7/docs/technotes/guides/visualvm/applications_remote.html). The only way to use it remotely would be to export DISPLAY to another machine.

Once you've connected to the JVM, click on Sampler and click the CPU button to start sampling. By default, when VisualVM samples a stack, it will skip over stack frames in the packages java.*, javax.*, sun.*, sunw.*, and com.sun.* (https://blogs.oracle.com/nbprofiler/entry/profiling_with_visualvm_part_2). You can setup a profiling preset under Tools > Options and specify a blank string for the "Do not profile packages" textbox to override this.

It does not appear that VisualVM data can be cropped to a particular time period, making it difficult to use for problems during a specific period.

VisualVM supports a plugin that can track DirectByteBuffer usage: https://blogs.oracle.com/nbprofiler/entry/new_memory_buffer_plugin_for
Tracing Profilers

IBM Java -Xtrace

For simple tracing profiler usage on IBM Java, -Xtrace is very easy to use: http://www-01.ibm.com/support/docview.wss?uid=swg21657391

Rational Application Developer (RAD) Profiler

The Rational Application Developer profiling platform provides three different analyses of application behavior:

- Memory-usage analysis
- Method-level execution analysis
- Thread analysis

Suited when you have an eclipse based IDE installed and are using RAD to develop software.

Profiling agents are executed alongside the JVM (and inside the JVM process) when that JVM is run with special JVMTI-specific VM arguments. When the profiling agents run, they collect data from the JVM in the form of execution, heap, or thread events. Within the Rational Application Developer profiling sphere, these agents are referred to as the Execution Analysis, Heap Analysis, and Thread Analysis agents.

The following tutorial walks you through how to go about setting up the profiling agent and collecting data


You can also profile outside of RAD using the standalone mode.

Rational Agent Controller (RAC)

The Rational Agent Controller (RAC) has a really broad set of supported operating systems: AIX, Linux, Linux s/390 (zLinux), Windows, Solaris, and z/OS: http://www-01.ibm.com/support/docview.wss?uid=swg27013420#v8. Once you've got the agent controller installed and the JVM instrumented, you can either gather data in headless mode which you load into Rational Application Developer, or start/pause monitoring remotely from RAD.

The RAC comes with a JVMTI profiling agent which has to be attached to the JVM. This profiler has a lot of native components which makes this a bit tricky. First, you'll need to add a generic JVM argument, such as:

"-agentpath:/opt/IBM/SDP/AgentController/plugins/org.eclipse.tptp.javaprofiler/libJPIBootLoader.so=JPIAgent:server=controlled,format=binary,file=log.trcbin;CGProf"

Note that the argument has to be specified with double quotes to avoid any issues with the semicolon in the Linux launcher. So if you already had some arguments, such as -Xgcpolicy:gencon, then your final generic JVM arguments would be:

-Xgcpolicy:gencon "-
agentpath:/opt/IBM/SDP/AgentController/plugins/org.eclipse.tptp.javaprofiler/libJPIBootLoader.so=JPIAgent:server=controlled,format=binary,file=log.trcbin; CGProf"

Next, we need to tell Linux how to load native library dependencies for libJPIBootLoader.so. To do this, we need to tell WAS to start with a specific LD_LIBRARY_PATH environment variable. Envvars can be set through the Environment Entries option (http://www-01.ibm.com/support/docview.wss?uid=swg21254153):

Name = LD_LIBRARY_PATH
Value = /opt/IBM/SDP/AgentController/plugins/org.eclipse.tptp.javaprofiler/:/opt/IBM/SDP/AgentController/lib

WAS is smart enough to append the library path you specify to the library path that it needs itself.

Use the server=controlled option in which case the JVM will not start until RAD connects to it (http://www.eclipse.org/tptp/home/downloads/4.5.0/documents/installguide/agentcontroller_45/linux/getting_started.html#Java_15). The reason we did this was so that we can control what gets profiled, since we weren't interested in profiling JVM startup. This option is recommended over server=enabled for high volume profiling (http://www-01.ibm.com/support/docview.wss?uid=swg21414403). Here are the basic steps we followed:

1. Start the RAC agent (RAStart.sh) before launching the application server
2. Launch the application server (it will immediately enter a wait state)
4. In some versions of RAD, this will immediately start profiling, in which case you'll probably want to click Pause - the JVM will continue to start but profiling will not be active
5. When you're ready, resume the actual profiling and continue as long as necessary
6. You'll probably want to select the option in RAD to save the data to a local file for post-analysis in addition to streaming it into RAD itself

There is also the option of using server=standalone which writes the profiling data to a local file and avoids the RAC itself and needing to connect in remotely from RAD. However, startup may take very long and create a lot of data which would have been cumbersome to analyze.

There are many ways to analyze the captured data: http://www.ibm.com/developerworks/rational/tutorials/profilingjavaapplicationsusingrad/index.html

For example, capture top -b -H -d 1800 -p $PID to gather accumulated CPU time per thread at the start of profiling and at the end and take the difference to find the threads that accumulated CPU and sort by that number. Next, within RAD's Execution Time Analysis, select the Call Tree tab and find these threads. Expand the threads and follow down the largest paths of cumulative time. Note that there may be some rows with very large cumulative times that are probably just the frames of the thread that are "waiting for work," such as a call to getTask or await, and these can be disregarded.
Once you find a high level method of interest (the art of profiling!), right click it and select Show Method Invocation Details. In the third table, "Selected Method Invokes," sort by Cumulative CPU Time, descending (if you don't have this column, you will need to make sure you have this option selected in one of the RAD attach/profiling screens when starting to profile). This will give the accumulated CPU time from a high level. You can then "drill down" further if you'd like to by doing the same procedure with rows from this table.

Note: Cumulative CPU time in the method invocation details is for the whole tracing profile, not from within the context of the call tree thread stack that you get here from.

**JInsight**

JInsight has two modes: profiling mode and cpuMode. Profiling mode is JInsight's primary raison d'être to do a full profile of all method executions on a JVM. This is extremely heavy and generally done with only one or a few concurrent transactions. cpuMode can gather timing statistics of method calls based on a configuration file, both in cpu microseconds consumed as well as elapsed times.

JInsight supports both JVMPI (-Xrun) and JVMTI (-agent*), although we will focus on JVMTI as JVMPI is deprecated. JVMPI required the JVM to call the agent on every event such as very frequent and basic operations like Object.hashcode() entry/exit. JVMTI however requires the agent to manipulate the Java Bytecodes of classes as they are loaded and insert Java instructions to callback the agent for whatever reason and at whatever point the agent requires.

**Profiling Mode**

Examples:
$ java -agentlib:jinsight$LIB=localFileName=/var/tmp/trace%p-%d.trc,localID=1234
$ java -agentpath:/opt/jinsight/libjinsight$LIB=localFileName=/var/tmp/trace%p-%d.trc,localID=1234

Start/stop (can be done multiple times per run)

$ jinctl-$BITMODE start 1234
$ jinctl-$BITMODE stop 1234

Name=LIBPATH
Value=/software/.../bin
The Value is the directory containing the JInsight .so file

Analysis
A vertical bar is a method execution. The space between two vertical bars represents the left method executing the right method. Clicking on the vertical bar (method), highlights all calls to methods of the class of that method.
Clicking on the space between two vertical bars highlights only all of those specific method calls of the right vertical bar.
If you zoom in far enough then method names will be shown instead of having to hover over.
Options > Uncheck "Short Names" to get full class names

If using profiler mode, gather multiple transactions and ensure hotspots are in both, because if a GC runs during some method, the GC wall clock time will be attributed to that method, so the only way to know something is heavy but non-GC, find it in multiple transactions.
Look at relative "intensity" in each thread – both stack depth and execution time. Note that the Y-axis is wall clock time. A method spanning some time X did not necessarily use X seconds of CPU (in fact, most likely not). Threads may be context switched by higher priority work, waiting for I/O, or the end of the method was not captured (e.g. at the end of captures).
Consider running 1 burst with 1 transaction, then another burst with 3 back-to-back, then another with 5 with time in between. This will help in visualizing the transactions from the background noise.
Questions to ask: Do we need to be doing something (or so often)?
Get a baseline without any transactions. Don't assume the transactions are doing all the work.
If a method name is not in the Execution profile, you can still find its signature (if it was seen [check Methods table]) by opening the profile data file in a binary text viewer and searching for the method name – the parameters should follow.

Keys
- Pan: Arrow keys, Page Up, Page Down
- Pan with mouse: Ctrl + Hold down left mouse button and move
- Zoom Home: Home key
• Compress time: <
• Expand time: >
• When you Ctrl+C to copy to clipboard, whatever is under the mouse at the time gets copied (not necessarily what's selected)

**cpuMode**

Examples:

```
$ java -agentlib:jinsight$LIB=
cpuMode,configFile=/var/tmp/jinsight/jinsight.cfg,logFile=/var/tmp/jinsight/trace.xml

$ java
-agentpath:/opt/jinsight/libjinsight$LIB=cpuMode,configFile=/var/tmp/jinsight/jinsight.cfg,logFile=/var/tmp/jinsight/trace.xml
```

Example cpuMode configuration file:

```xml
transaction com/ibm/ws/proxy/channel/http/HttpProxyConnectionLink
processResponseWork(Lcom/ibm/wsspi/channel/framework/VirtualConnection;Lcom/ibm/wsspi
/http/channel/error/HttpError;)V WriteData
fragment com/ibm/ws/proxy/log/http/HttpProxyLogImpl logNCSACommon
(Lcom/ibm/wsspi/proxy/filter/http/HttpProxyServiceContext;Lcom/ibm/wsspi/proxy/log/LogLev
el;;)V NCSALog
```

Example output from cpuMode:

```
<transaction identity="WriteData" cpuMicros="2485" elapsedMicros="3190"
thread="WebSphere t=009b0a60" seq="13709" startTime="Mon Sep 27 15:26:28 2010">
<fragment identity="NCSALog" cpuMicros="1228" elapsedMicros"1459" seq="13710"
startTime="Mon Sep 27 15:26:28 2010">
</fragment>
</transaction>
```

For cpuMode on z/OS, the config file should be uploaded as ASCII (in FTP, ASC) and written in an editor to enforce Unix style line endings (not Windows).

You can get what to put in the config file from the Execution view by hovering over a method call and pressing Ctrl + C, and then taking everything after the "forConfig > " text.

**Performance Inspector**

Performance inspector is a suite of profilers including sampling and tracing profilers and other tools for various operating systems. An open source version still exists but it is not actively maintained: [http://perfinsp.sourceforge.net/](http://perfinsp.sourceforge.net/)

**WebSphere Application Server**

WebSphere Application Server (WAS) is a Java Enterprise Edition application server with functions such as serving websites.
WAS Traditional is the name of the original product produced for over 15 years and still a strategic offering. WAS Liberty is a partial rewrite of WAS Traditional which is lighter-weight, more flexible, etc. How to choose: http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/wasdev/documentation/ChoosingWASClassicorWASLiberty.pdf

**WebSphere Application Server Recipe**

1. Upgrade to the latest version and fixpack of WAS as it has a history of making performance improvements over time.

Additionally, see the chapters for WAS Traditional or WAS Liberty.

**Benchmarks**

The benchmark results as of February 2015 for the SPECjEnterprise2010 benchmark suite show that traditional WAS 8.5.5.4 outperforms Oracle WebLogic 12.1.3 by 32% on a per core basis and 3% on a per processor basis, both on the latest Intel Haswell EP Processors:
WAS shows consistent performance gains across releases:

**WebSphere Application Server Traditional**

WAS Traditional is also known as the WAS Full Profile, traditional WAS (tWAS), or WAS Classic.


**WAS Traditional Recipe**

In addition to the [overall recipe in the WebSphere Application Server chapter](http://www-01.ibm.com/support/knowledgecenter/SSAW57/mapfiles/product_welcome_wasnd.html),
1. Use the Performance Monitoring Infrastructure (PMI) to monitor various statistics such as thread pool usage, database connection pools, etc. Use a tool such as IBM ITCAM, IBM Tivoli Performance Viewer (TPV) in the Administrative Console, the WAS Performance Tuning Toolkit, etc.
2. Thread pools should not be consistently saturated.
3. Database connection pools should not be consistently saturated.
4. Monitor response times of key application components (e.g. servlets, databases, MDBs, etc.).
5. Apply the production performance tuning template.
6. On WAS 8.5, explicitly install and switch to the most recent version of Java available using managelsdk.
7. Switch to High Performance Extensible Logging (HPEL) and disable JMX log notifications (-Dcom.ibm.ejs.ras.disablerasnotifications=true).
8. Review SystemOut/SystemErr/HPEL, FFDC and application logs for any errors, warnings, or high volumes of messages.
9. If possible, configure and use servlet caching/Dynacache.
10. Don't neglect to monitor and tune the node agents and deployment manager (particularly garbage collection).
11. Ensure that when WAS fixpacks have been applied, the correct service release of Java was also upgraded: https://www-304.ibm.com/support/docview.wss?uid=swg27005002
12. If Single Sign On (SSO) is enabled, test whether performance is better with web inbound attribute propagation enabled (default) or disabled.

**WAS Basics**

"In general, a large number of applications will realize some improvement from tuning in three core areas: the JVM, thread pools, and connection pools."


Review all messages in SystemOut.log, SystemErr.log (or HPEL logs), native_stdout.log, native_stderr.log, application logs (such as log4j), and First Failure Data Capture (FFDC) logs. Note that with FFDC logs, an exception will often only create an FFDC stack and information file on the first occurrence (this is the design of FFDC), but you can review the _exception.log summary file for the number of times that exception was thrown.

Review the WAS logs and eliminate (or try to reduce) any warnings and exceptions. If customers say, "Oh, those warnings/errors are 'normal'," persist in investigating them anyway and pushing for them to be eliminated. Imagine you are tuning a sports car for optimal performance and there's a warning light in the dashboard. Yes, it is possible that the warning is "normal" and will not impact performance, but unless you have direct evidence that this is so, you should go under the assumption that such warnings and errors are signs of potential performance problems. You should resolve any warnings that the designers of the car thought worthy of highlighting. Such warnings may have indirect or subtle performance impacts that may not be easy to theoretically understand. At minimum, the system is spending resources tracking and reacting to these warning conditions. In the case of exceptions, these include stack traces which may cost a significant amount to create, even if an exception is caught and suppressed.

Continue to monitor for warnings and errors during performance runs, particularly hung thread warnings (WSVR0605W) and CPU starvation warnings (HMGR0152W).
If you don't know the host names and ports of the various nodes and servers but you have access to the configuration, consult the file `${WAS}/profiles/${PROFILE}/config/cells/${CELL}/nodes/${NODE}/serverindex.xml` and search for the relevant virtual hosts such as WC_adminhost_secure and WC_defaulthost. The administrative server is normally at https://${DMGRHOST}:${DMGRADMINHOST_SECUREPORT}/admin


Key administrative concepts:

- An installation of WAS Network Deployment has a set of 0 or more profiles which represent nodes. These nodes share the same runtime binaries. An installation is either 32-bit or 64-bit.
- A profile/node is a set of 0 or more managed servers (most commonly, application servers). The node has a process called the node agent which manages the configuration of the servers, and may also orchestrate their runtime behavior (starting and stopping them, restarting failed application servers, etc.).
- A special type of profile/node is the deployment manager profile (dmgr). This profile represents a cell, which is a set of 1 or more nodes, including the deployment manager node/server itself. The deployment manager holds the primary configuration data for the whole cell and the deployment manager runs the Administrative Console application which is used to administer the whole cell.
- To participate in a cell, a node must be federated with the deployment manager profile using tooling or the "addNode" command from the newly created profile's bin directory.
- Usually, WAS will be installed on one physical/virtual machine with a single deployment manager profile/node, and then a set of 1 or more other physical/virtual machines will install WAS with a single profile/node (representing that machine) federated into the deployment manager.
- When a configuration change is made in the deployment manager, nodes must be synchronized with the deployment manager to get the updated changes. By default, this happens automatically every 1 minute (see the node agent's file synchronization service settings). Nodes can also be manually synchronized while saving the changes into the DMGR (click Review, and then check the box to synchronize the nodes), or through the Nodes collection under System Administration. For example, if you have 10 machines, with 5 application servers each, you install an application into the deployment manager, then when the nodes are synchronized, the application will be distributed to all the nodes and then it can be started across all of those servers.

**Performance Tuning Templates**

WAS ships with a set of tuning templates including a production tuning template for a typical production environment. The script is found in `${WAS}/bin/applyPerfTuningTemplate.py` and the production template is found in `${WAS}/scriptLibraries/perfTuning/V70/peak.props`.

The production tuning template applies the following changes ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tuneappserv_script.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tuneappserv_script.html)); however, we recommend re-enabling PMI after the script completes.
Note: Although the production tuning template is in a folder named V70, it applies to later versions of WAS.

Here is a Unix example of running the production tuning template on one application server. This assumes that the deployment manager and node agent are started.

```bash
$ cd ${WAS}/bin/
$ ./wsadmin.sh -lang jython -f applyPerfTuningTemplate.py -nodeName node1 -serverName server1 -templateFile ../scriptLibraries/perfTuning/V70/peak.props
```

**General Tuning**

Check "Start components as needed" to potentially improve startup time by not starting components of the application server that are not used


Tune the XML parser definitions by updating the jaxp.properties and xerces.properties files in the `${app_server_root}/jre/lib` and adding:

```properties
javax.xml.parsers.SAXParserFactory=org.apache.xerces.jaxp.SAXParserFactoryImpl
org.apache.xerces.xni.parser.XMLParserConfiguration=org.apache.xerces.parsers.XIncludeAwareParserConfiguration
```


**Shared Libraries**

Use shared libraries where possible to reduce memory usage.

**Change Java Software Development Kit (SDK)**

In recent versions of WAS, use the managesdk command to change the Java Software Development Kit:


Example: List available SDKs:

```bash
$ ./managesdk.sh -listAvailable
CWSDK1003I: Available SDKs:
    CWSDK1005I: SDK name: 1.7_64
    CWSDK1005I: SDK name: 1.6_64
```

Example: Change SDK for all profiles:
$ ./managesdk.sh -enableProfileAll -sdkName 1.7_64
CWSDK1017I: Profile AppSrv01 now enabled to use SDK 1.7_64.

Scaling and Large Topologies

Scaling and Large Topologies Recipe

1. Use clusters to scale horizontally and vertically, and to support failover and easier administration. If using WAS >= 8.5, consider using dynamic clusters.
   • Very large topologies also employ multiple cells for the same application(s). This allows for deployment of new application versions or configurations to only one of the cells; if the change breaks, it affects only that cell. Multiple cells can be problematic if significant database schema changes are made.
2. If using the High Availability Manager or any functions that require it (e.g. EJB WLM, SIB, etc.):
   • Processes such as application servers and node agents must be in the same core group, or part of bridged core groups.
   • In general, the number of processes in a single core group should not exceed 200. Practically, this number is limited by the CPU usage, heartbeat intervals, and number of available sockets.
   • The members of a core group should be on the same LAN.
   • The members of a cell should not communicate with one another across firewalls as that provides no meaningful additional security and complicates administration.
   • Create dedicated preferred coordinators for a core group with a large default maximum heap size (e.g. -Xmx1g).
   • If using core group bridges, create dedicated bridge servers with a large default maximum heap size (e.g. -Xmx1g).
   • Start or stop groups of processes at the same time to reduce the effects of view changes.
   • Change the HAM protocols to the latest versions: IBM_CS_WIRE_FORMAT_VERSION and IBM_CS_HAM_PROTOCOL_VERSION
3. If you are not using the High Availability Manager, it is not recommended to disable it, but instead to create multiple cells or bridged core groups.

Clusters

Clusters are sets of servers that are managed together and participate in workload management. Clusters enable enterprise applications to scale beyond the amount of throughput capable of being achieved with a single application server. Clusters also enable enterprise applications to be highly available because requests are automatically routed to the running servers in the event of a failure. The servers that are members of a cluster can be on different host machines.... A cell can include no clusters, one cluster, or multiple clusters.

Servers that belong to a cluster are members of that cluster set and must all have identical application components deployed on them. Other than the applications configured to run on them, cluster members do not have to share any other configuration data. One cluster
A vertical cluster has cluster members on the same node, or physical machine. A horizontal cluster has cluster members on multiple nodes across many machines in a cell. You can configure either type of cluster, or have a combination of vertical and horizontal clusters.

http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-mp&topic=crun_srvgrp

**Dynamic Clusters**

WAS 8.5 includes Intelligent Management which provides dynamic clusters. Dynamic clusters provide the same functionality of traditional clusters and more. See the [Intelligent Management section](http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-mp&topic=crun_srvgrp).

**Large Topologies, High Availability Manager**

The latest guidance on core group size is: "Core groups containing more than 100 member should work without issue in many topologies. Exceeding a core group of 200 members is not recommended." ([http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_cgscale.html](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_cgscale.html)). If the size of your core group is too large, consider core group bridging: ([http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_coregroupbridge.html](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_coregroupbridge.html)).

It is a best practice to use the newer High Availability Manager (HAManager) protocols, particularly with large topologies ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/crun_ha_protocol_ver.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/crun_ha_protocol_ver.html)):

- IBM_CS_WIRE_FORMAT_VERSION=6.1.0
- IBM_CS_HAM_PROTOCOL_VERSION=6.0.2.31

In general, for small core groups, HA/DCS issues are just symptoms of other issues like CPU exhaustion, network instability, etc. Some other things to consider:

1. Set a preferred coordinator: "Remember that coordinator election occurs whenever the view changes. Electing a new coordinator uses a lot of resources because this process causes increased network traffic and CPU consumption. Specifying a preferred coordinator server, whenever practical, helps eliminate the need to make frequent coordinator changes... Preferred coordinator servers should be core group processes that are cycled as infrequently as possible. The preferred coordinator servers should also be hosted on machines with excess capacity." ([http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_coordinator.html?lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_coordinator.html?lang=en)) and "Even though it is possible to use a deployment manager as a core group coordinator, it is recommended that you use an application server that is not a deployment manager." ([http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_coordinator.html?lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/crun_ha_coordinator.html?lang=en))

If using core group bridges:

1. Core group bridges be configured in their own dedicated server process, and that these processes have their monitoring policy set for automatic restart.
2. For each of your core groups, you set the IBM_CS_WIRE_FORMAT_VERSION core group custom property to the highest value that is supported on your environment.
3. To conserve resources, do not create more than two core group bridge interfaces when you define a core group access point. You can use one interface for workload purposes and another interface for high availability. Ensure that these interfaces are on different nodes for high availability purposes. For more information, see the frequently asked question information on core group bridges.
4. You should typically specify ONLY two bridge interfaces per core group. Having at least two bridge interfaces is necessary for high availability. Having more than two bridge interfaces adds unnecessary overhead in memory and CPU.


**Large Topology Theory**

The WebSphere Application Server Network Deployment product is tuned for small to modest-sized cells in its default configuration. By understanding how the application server components are designed and behave, it is possible to tune the product so that large topologies, which contain hundreds of application servers, can be created and supported.

The primary thing that limits the size of the cell is the need to support shared information across all or a large set of application server processes. The breadth and currency requirements for shared information, which is something that must be known by all or many application server instances within the cell, present a challenge for any distributed computing system.

An instance of the High Availability Manager (HAManager) runs inside every process in a Network Deployment cell, including the deployment manager, node agents, application servers and proxy servers. The HAManager provides a set of frameworks and facilities that other WebSphere services and components use to make themselves highly available.

The HAManager relies on core groups. A core group is a collection of firmly coupled processes which collaborate to form a distributed group communication service. It is a requirement that all members of a core group must be in the same cell.

As the size of a cell increases, it may be necessary to partition the cell into multiple core groups, because core groups do not scale to the same degree as other cell constructs. When a cell has been partitioned, it is often necessary to share routing information between core
groups. For example, a web application located in core group 1 may call an enterprise bean application located in core group 2. There are also cases where it is necessary to share routing information across cells. A Core Group Bridge provides this capability to extend the HAManager bulletin board beyond core group boundaries. Core groups that are connected with a core group bridge can share routing data.

While there are no WebSphere-defined limits on the size of a core group, there are practical limits. The practical limits are primarily driven by available resources and stability. The amount of resource used by the HAManager and core groups depends on a number of factors, including the core group size, core group configuration settings, the amount of routing data required to support the deployed applications, and quality of service settings.

All members of a core group must be located on machines that are connected by a high speed local area network (LAN). Do not locate members of the same core group on machines that are connected by a wide-area network (WAN). Do not place members of a cell across a firewall, as a firewall provides no meaningful security between members of WebSphere processes.

For active heart-beating, the default configuration settings provide a 30 second heartbeat interval and a 180 second heartbeat timeout, meaning that failovers initiated by the active failure detection mechanism take longer than failovers initiated by socket closing events. This default setting represents a compromise between failover time and background CPU usage. If faster failover is required, then the configured heartbeat timeout can be lowered, at the cost of additional background CPU usage.

The amount of background CPU used by the HAManager for heart-beating and failure detection is affected by the heartbeat interval and core group size. Starting with a core group of 100 members as a baseline using the default heartbeat interval of 30 seconds, approximately 20% of the background CPU used by a WebSphere product application server at idle is spent on heartbeat processing.

Observing a high background CPU at idle can be indicative of the core group (or groups) approaching the practical limit for your infrastructure and deployment. If you encounter high idle CPU, you should explore decreasing the number of members in existing core groups by moving processes to a new bridged core group to reduce the background CPU.

It is a best practice to configure one or more preferred coordinator processes for each core group. This limits the movement of the coordinator and number of state rebuilds. Ideally, assign processes that do not host applications and are located on machines with spare capacity as preferred coordinators.

In a topology that contains core group bridges, it is a best practice to create stand-alone application server processes that do not host applications to function as both bridge interfaces and preferred coordinators.

The limits on the size of a core group are practical, not programmatic. The most important considerations in determining core group sizes are resource usage and stability.
The HAManager uses CPU, memory, and network resources. Generally speaking, memory is not a major factor in determining core group size. The amount of long-term heap memory required for routing data is determined by the topology and applications installed, not by the core group size. Splitting a cell into multiple core groups does not reduce the memory required for the routing data. Therefore, the size of the core group is determined almost exclusively based on the CPU required to establish and maintain the group communication service.

The HAManager uses CPU to establish network connections and group communication protocols between running members of the core group. As processes are started, connections are opened to other core group members and the group membership and communication protocols are updated to include the newly started members in the group, or “View”. This change is often referred to as a “View Change.” As processes are stopped, connections are closed and the group membership and communication protocols are updated to exclude the stopped members.

Therefore, starting or stopping a process causes the HAManager to use CPU to open or close connections and update the group communication service. This means that starting or stopping one process causes some CPU usage by all other running core group members. As the size of the core group grows, the number of connections and size of the group membership will grow, meaning that more CPU will be used for large core groups than for small ones. There is also some short-term usage of heap memory to send the network messages required to update the group communication service.

In general, it is more efficient to start or stop groups of processes at the same time, allowing the HAManager to efficiently consolidate multiple group membership and communication protocol changes within a single view change.

An additional factor to consider is the number of sockets that are consumed to create the connections between core group members. The members of a core group form a fully connected network mesh, meaning every member connects directly to every other member. The total number of sockets used to connect all members of a core group approaches $n^2$, where $n$ is the number of core group members. Suppose for example that you tried to create a core group of 200 members on a single machine. The number of sockets required would be 200 x 199 or 39,800 sockets. The same 200 members split into 4 core groups of 50 members each would require 4 x 50 x 49 or 9800 sockets.

Core groups containing more than 100 members should work without issue in many topologies. Exceeding a core group size of 200 members is not recommended.

Important: Disabling the HAManager might cause some critical functions to fail.

For the reasons outlined previously, rather than disabling HAManager, either create multiple cells or partition the cell into multiple core groups and create bridges. Even if you do not currently use a component that requires HAManger, you may require one at a later time.
IBM_CS_DATASTACK_MEG

In recent versions of WAS, the default values of IBM_CS_DATASTACK_MEG and the transport buffer size are usually sufficient. (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/urun_ha_cg_custprop.html and https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/trun_ha_cfg_replication.html)

Setting the two memory sizes does not increase the amount of static heap allocated by the HAManager. These settings affect flow control (how many messages are allowed to pass through the HAManager at any one point in time before we stop sending messages). Larger settings allow more efficient communications. We have seen situations (on large topologies) where having the memory sizes set too small will lead to problems. Generally speaking, the messages have already been allocated by the time they reach the congestion checker so this doesn't give us much relief on the heap issues... increasing the memory sizes has only helped from a stability standpoint.
HAManager Architecture

WAS Performance Monitoring

WAS provides two broad capabilities for performance monitoring: the Performance Monitoring Infrastructure (PMI), and Application Response Measurement (ARM). PMI is a statistical sampler inside WAS that periodically gathers averaged or instantaneous data on various components such as HTTP response times, thread pools, database connection pools, messaging, etc. Application Response Measurement, also called Request Metrics, traces and times individual requests as they execute through various components. This difference in approaches was covered previously in the Java Profilers chapter. To recap, statistical samplers (e.g. PMI) are very lightweight and help resolve the majority of issues; however, they will not capture data on particular requests. Whereas, tracing profilers (e.g. ARM) are much heavier weight and more complex to analyze. PMI exposes its data through JMX and ARM exposes its data either through log files or a Java agent API.
Performance Monitoring Infrastructure (PMI)

What is PMI?
The Performance Monitoring Infrastructure (PMI) uses a client-server architecture. The server collects performance data from various WebSphere Application Server components. A client retrieves performance data from one or more servers and processes the data. This data consists of counters such as servlet response time and database connection pool usage.

PMI supports five sets of counters: None, Basic, Extended, All, and Custom. The Basic set is enabled by default and has an overhead of approximately 2%, whether or not it is actively being logged or queried:

"Basic overhead ~= 2%
Extended overhead ~= 3%
Custom ~= 2% - 6%"

http://www-01.ibm.com/support/docview.wss?uid=swg21206317

In general, it is recommended to run with PMI enabled, even in production. Running without PMI is equivalent to flying a plane without instruments. However, for the purposes of a benchmark, after you've "completed" your tuning, for the final run you may consider reducing or disabling PMI. Disabling PMI completely may cause a small throughput improvement. In one benchmark, about 2%

Various Dimensions of Monitoring

It is useful to conceptualize different PMI statistics into groups. The first dimension is the "end user view" or a black box view of your application. This gives you a view as to how the application is performing and what are the response times taken to serve the requests. For example, for HTTP requests, the PMI counters are Web Applications/ServiceTime.

The second dimension is the "resources utilization view" of the system involved in the user activity. This will tell you the basic health of your system, including CPU, memory consumption, JVM health, as well as the health of various resources available such as HTTP sessions, connection pools, thread pools, etc. This dimension corresponds to the "what resource is constrained" portion of the problem diagnosis. For example, for HTTP requests, the PMI counters are Thread Pools/ActiveCount and JDBC Connection Pools/FreePoolSize, as well as JVM Runtime/ProcessCPUUsage and JVM Runtime/UsedMemory.

The third dimension is the "application view." Application code typically runs as a servlet or enterprise java bean to access various back-ends such as databases, web services, etc. For example, for HTTP requests, the PMI counters are Enterprise Beans/MethodResponseTime.

The data points are then retrieved using a web client, Java client or JMX client. WebSphere Application Server provides the built-in Tivoli Performance Viewer (TPV), which is embedded into WAS admin console.

Tivoli Performance Viewer (TPV)
The Tivoli Performance Viewer (TPV) retrieves performance data by periodically polling the PMI
service of the application server that is being monitored. TPV is not part of any external Tivoli tool. TPV is part of the WebSphere Application Server administrative console.

To minimize the performance impact, Tivoli Performance Viewer polls the server with the PMI data at an interval set by the user. All data manipulations are done in the Tivoli Performance Viewer. The Tivoli Performance Viewer’s GUI provides controls that enable you to choose the particular resources and counters to include in the view and whether to visualize in chart or table form.

In a Network Deployment environment, the node agent maintains each monitored server's per-user buffer. When the TPV monitor is enabled in the administrative console, the deployment manager polls the node agents for data to display. Therefore, it's important to monitor the performance of the deployment manager and node agents themselves when using PMI and/or TPV, particularly verbose garbage collection. There will be some additional overhead when enabling TPV, but mostly in the node agents and particularly in the deployment manager.

In the administrative console, select Monitoring and Tuning > Performance Viewer > Current activity, the check the box next to “server1” and click "Start Monitoring." After that operation completes, click the link on an application server:
The performance data for this server.
Welcome
- Server
- Applications
- Resources
- Security
- Environment
- System administration
- Monitoring and Tuning
  - Performance Monitoring Infrastructure (PMI)
  - Request Metrics
  - Performance Viewer
    - Current Activity
    - View Logs
- Troubleshooting
- Service integration
- UDDI

Tivoli Performance Viewer > server1
The performance data for this server.

+ SIB Service
  - Enterprise Beans
    - trade#TradeEJB
  - Message Drives
  - Stateless Session
    - TradeEJB
    - KeySequence
  - Entity Bean
+ Dynamic Caching
  - JDBC Connection Pools
    - Cloudscape JDBC
  - DB2 Universal JDBC
+ HAManager
  - JCA Connection Pool
  - JVM Runtime
+ Object Pool
+ ORB
+ Servlet Session Manager
+ Thread Pools
+ Transaction Manager
+ Web Applications
+ Web Services
+ Workload Manager
+ Web services Gateway
In the table under the graph, check the boxes next to the line plots that should be drawn in the graph. For example, you may want to check the JDBC "UseTime" and "WaitTime" to review database response times and times to acquire a database connection, respectively. By default, the graph updates every 30 seconds. The "Value" and "Scale Value" columns display the last value of the counter (the Scale Value is used for the graph). In the following example, the average JDBC use time of a connection is 18.5 milliseconds, the average response time of all servlets is 1.85 milliseconds, and the average concurrently active threads in all WAS thread pool is 8.
The modules may be further broken down in detail. For example, you may check each servlet under Web Applications, click View Modules, and review the average response time per servlet:

<table>
<thead>
<tr>
<th>jdbc/TradeDataSource</th>
<th>CreateCount</th>
<th>10.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CloseCount</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>PoolSize</td>
<td>10.0</td>
</tr>
<tr>
<td></td>
<td>FreePoolSize</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>WaitingThreadCount</td>
<td>21.0</td>
</tr>
<tr>
<td></td>
<td>PercentUsed</td>
<td>100.0</td>
</tr>
<tr>
<td></td>
<td>UseTime</td>
<td>26.132856</td>
</tr>
<tr>
<td></td>
<td>WaitTime</td>
<td>65.49429</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WebContainer</th>
<th>ActiveCount</th>
<th>35.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PoolSize</td>
<td>50.0</td>
</tr>
<tr>
<td></td>
<td>DeclaredThreadHungCount</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>ClearedThreadHangCount</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>ConcurrentHungThreadCount</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Web Applications</th>
<th>RequestCount</th>
<th>124669.0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ServiceTime</td>
<td>22.386284</td>
</tr>
</tbody>
</table>
For more details, please visit the following URLs.


### What metrics should you gather?

PMI Basic includes the following counters. The most commonly useful are highlighted in **bold**:

- Enterprise Beans.CreateCount
- Enterprise Beans.RemoveCount
- Enterprise Beans.ReadyCount
- **Enterprise Beans.MethodCallCount**: The number of calls to the business methods of the bean.
- **Enterprise Beans.MethodResponseTime**: The average response time in milliseconds on the business methods of the bean.
- Enterprise Beans.PooledCount
- **Enterprise Beans.MessageCount**: MDB: The number of messages delivered to the onMessage method of the bean.
- Enterprise Beans.PassiveCount
- Enterprise Beans.MethodReadyCount
- Enterprise Beans.ReadLockTime
- Enterprise Beans.WriteLockTime
- Enterprise Beans.LockCancelCount
- Enterprise Beans.AsyncWaitTime
- Enterprise Beans.AsyncQSize
- Enterprise Beans.AsyncCancelCount
• Enterprise Beans.AsyncFNFFailCount
• Enterprise Beans.AsyncFutureObjectCount
• Enterprise Beans.Discards
• JDBC Connection Pools.CreateCount
• JDBC Connection Pools.CloseCount
• JDBC Connection Pools.PoolSize
• JDBC Connection Pools.FreePoolSize: The number of free connections in the pool.
• JDBC Connection Pools.WaitingThreadCount
• JDBC Connection Pools.PercentUsed
• JDBC Connection Pools.UseTime: The average time a connection is used... Difference between the time at which the connection is allocated and returned. This value includes the JDBC operation time.
• JDBC Connection Pools.WaitTime: The average waiting time in milliseconds until a connection is granted.
• JVM Runtime.HeapSize
• JVM Runtime.UsedMemory: The used memory in the JVM run time.
• JVM Runtime.UpTime
• JVM Runtime.ProcessCpuUsage: The CPU Usage (in percent) of the Java virtual machine.
• JCA Connection Pools.CreateCount
• JCA Connection Pools.CloseCount
• JCA Connection Pools.PoolSize
• JCA Connection Pools.FreePoolSize: The number of free connections in the pool.
• JCA Connection Pools.WaitingThreadCount
• JCA Connection Pools.UseTime: Average time in milliseconds that connections are in use.
• JCA Connection Pools.WaitTime: The average waiting time in milliseconds until a connection is granted.
• Servlet Session Manager.LiveCount: The number of local sessions that are currently cached in memory from the time at which this metric is enabled.
• System Data.CPUUsageSinceLastMeasurement: The average system CPU utilization taken over the time interval since the last reading... On SMP machines, the value returned is the utilization averaged over all CPUs.
• Thread Pools.ActiveCount: The number of concurrently active threads. Note: The ActiveCount value can include a count for a long-running thread that is used for asynchronous I/O. Under these circumstances, it is possible that even when there is no apparent activity on the thread pool, the ActiveCount value will never reach zero.
• Thread Pools.PoolSize
• Transaction Manager.ActiveCount
• Transaction Manager.CommittedCount
• Transaction Manager.RollbackCount: The total number of global transactions rolled back.
• Web Applications.RequestCount: The total number of requests that a servlet processed.
• Web Applications.ServiceTime: The response time, in milliseconds, of a servlet request.

Warning: If you are using a generational garbage collection policy such as the IBM gencon or balanced policies (gencon is the new default starting in WAS version 8), or most of the Oracle policies, then be aware that the JVM Runtime.UsedMemory statistic may be deceiving because it is sampling based on time rather than global collections, so samples may report high memory utilization that may consist of a lot of trash that will be cleaned up at the next global collection. Use verbose garbage collection
instead.

We do not cover the PMI Extended set because we recommend that if you do plan on doing complex PMI analysis, that you should use the Custom set instead.

In general, we recommend the PMI Custom set with all of the applicable highlighted counters above as well as the following counters (where applicable):

- **Dynamic Caching.HitsInMemoryCount**: The number of requests for cacheable objects that are served from memory. For servlet instance, locate it under template group. For object instance, locate it under object group.
- **Dynamic Caching.MissCount**: The number of requests for cacheable objects that were not found in the cache. For servlet instance, locate it under template group. For object instance, locate it under object group.
- **JDBC Connection Pools.JDBCTime**: The amount of time in milliseconds spent running in the JDBC driver which includes time spent in the JDBC driver, network, and database.
- **JDBC Connection Pools.PrepStmtCacheDiscardCount**: The total number of statements discarded by the least recently used (LRU) algorithm of the statement cache.
- **Mediation.MediatedMessageCount**: The number of messages that have been mediated at a mediated destination.
- **Mediation.MediationTime**: The amount of time in milliseconds taken to mediate a message at a mediated destination.
- **MEStats.BufferedReadBytes**: Number of bytes of data that have been received from the network and are held pending further processing. Large values might indicate that the application server is unable to process data fast enough to keep up with the other application server processes hosting messaging engines.
- **MEStats.BufferedWriteBytes**: Number of bytes of data being held pending transmission. Large values might indicate network congestion or application server processes hosting messaging engines that are unable to process data fast enough to keep up with the application server.
- **QueueStats.AvailableMessageCount**: The number of messages available for a queue for consumption. If this number is close to the destination high messages value then review the high messages value.
- **QueueStats.LocalMessageWaitTime**: The time spent by messages on this queue at consumption. If this time is not what was expected then view the message in the administrative console to decide what action needs to be taken.
- **Servlet Session Manager.ExternalReadTime**: The time (milliseconds) taken in reading the session data from the persistent store. For multirow sessions, the metrics are for the attribute; for single row sessions, the metrics are for the entire session. Applicable only for persistent sessions. When using a JMS persistent store, you can choose to serialize the replicated data. If you choose not to serialize the data, the counter is not available.
- **Servlet Session Manager.ExternalWriteTime**: The time (milliseconds) taken to write the session data from the persistent store. Applicable only for (serialized) persistent sessions. Similar to external Read Time.
- **Servlet Session Manager.LifeTime**: The average session life time in milliseconds (time invalidated - time created)
- **Servlet Session Manager.NoRoomForNewSessionCount**: Applies only to sessions in memory with AllowOverflow=false. The number of times that a request for a new session cannot be handled because it exceeds the maximum session count.
- **Servlet Session Manager.SessionObjectSize**: High impact - debugging only: The size in bytes of
(the serializable attributes of ) in-memory sessions. Only session objects that contain at least one serializable attribute object is counted. A session can contain some attributes that are serializable and some that are not. The size in bytes is at a session level.

- **Servlet Session Manager.TimeoutInvalidationCount**: The number of sessions that are invalidated by timeout.
- **Thread Pools.ConcurrentHungThreadCount**: The number of concurrently hung threads
- **Web Applications.AsyncContext Response Time**: The response time (in milliseconds) for an AsyncContext associated with a servlet to complete.
- **Web Applications.ErrorCount**: Total number of errors in a servlet or JavaServer Page (JSP).
- **Web services.ProcessedRequestCount**: The number of requests the service successfully processed.
- **Web services.ResponseTime**: The average response time (in milliseconds) for a successful request

**Configuring Custom PMI**

In the WAS administrative console, navigate to Servers > Server Types > WebSphere Application Server > server1 > Performance Monitoring Infrastructure and click on "Custom." Click on the "Runtime" tab, and for example, expand "Servlet Session Manager," click on "DayTrader3#web.war," check "SessionObjectSize" and click "Enable."

**Application servers > server1 > Performance Monitoring Infrastructure (PMI) > Custom monitoring level**

Use this page to configure Performance Monitoring Infrastructure (PMI)

Logging TPV Data

TPV is a live view but for most system monitoring, problem analysis, or performance tuning, you will want to look at the data after the fact. TPV supports sending the data to log files and loading those files
into any administrative console for playback. TPV logging is a bit cumbersome because the log must be restarted after every application server restart; however, this can be automated with wsadmin scripts.

Logging TPV data in a production environment may have a significant overhead. Consider using a monitoring product such as ITCAM before trying to use TPV logging in production.

Select Monitoring and Tuning > Performance Viewer > Current activity, click the link on "server1," and click the "Log" link under settings in TPV:

```
server1
  + Advisor
  + Settings
  - Log
  + Summary Reports

Performance Modules
  + DCS Statistics
    + ExtensionRegistryStats
    - Security Authentication
  + SIB Service
  + Enterprise Beans
  + Dynamic Caching
  + JDBC Connection Pools
    + HAManager
    + JCA Connection Pools
    + JVM Runtime
    + Object Pool
```

Set "Duration" to 300000000, "Maximum File Size" to 100, "Maximum Number of Historical Files" to 5, "Log Output Format" to "Binary," click "Apply," and then click "View Modules."

Click the "Start Logging" button:

```
Start Logging
```

Files will be written to /opt/IBM/WebSphere/AppServer/profiles/AppSrv01/logs/tpv/, for example.

Later, when you want to view the logs, in the administrative console, select Monitoring and Tuning > Performance Viewer > View Logs, click "Browse," select
Check the performance modules as before, click View Modules, and use the backwards, stop, play, and forward buttons to review the collected data. By default, the log will be played back automatically.

Note: If there is a very short duration of data, you may not see all of the buttons above as all of the data can be displayed in one view.

**PMI Details**

In general, use ThreadPool.ActiveCount over ThreadPool.PoolSize, as the former is the average concurrently active threads in a thread pool, whereas the latter is simply the size of the thread pool. ActiveCount is an instantaneous measurement.

**Runtime Performance Advisors (RPA)**

Runtime Performance Advisors (RPAs) are pieces of code built into WAS that may be enabled to watch for certain performance issues and periodically report tuning recommendations. They are disabled by default:

Tuning WebSphere Application Server involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data, and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.

(https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc)
Note: The PMI service must be enabled for RPAs. If an RPA is enabled and the needed PMI counters are not already enabled, then the configuration will be updated to enable those counters.

An RPA runs in one of two places:

1. In each application server JVM and provides warning level advice in SystemOut/Runtime Messages/JMX (Performance and Diagnostic Advisor). Advice provided on: WebContainer and ORB thread pools, connection pool size, persisted session sizes and times, prepared statement cache size, session cache size, and memory leak detection.

2. In the node agent and provides advice in the administrative console Tivoli Performance Viewer (Tivoli Performance Viewer advisor). Advice provided on: WebContainer and ORB thread pools, connection pool size, persisted session sizes and times, prepared statement cache size, session cache size, dynamic cache size, and JVM heap size.

In general, JVM advisors are used to review advice after the fact, whereas TPV advisors are used when actively monitoring TPV data.

Warning: If you are using a generational garbage collection policy such as the IBM gencon or balanced policies (gencon is the new default starting in WAS version 8), or most of the Oracle policies, then be aware that the memory leak detection advice may report false positives. This is due to the fact that the advisor samples heap usage to minimize performance impact; however, the design of generational policies means that heap usage will show a leaking profile in between full garbage collections as the tenured regions fill up with garbage. Starting in WAS 8.5, instead of using the memory leak detection advice, you should use the excessive memory usage and excessive garbage collection health policies with usexdHeapModule=true. This has been resolved in APAR PI28801: [http://www-01.ibm.com/support/docview.wss?uid=swg1PI28801](http://www-01.ibm.com/support/docview.wss?uid=swg1PI28801)

Application Response Measurement (ARM) / Request Metrics

Request metrics is a tool that enables you to track individual transactions, recording the processing time in each of the major WebSphere® Application Server components... As a transaction flows through the system, request metrics includes additional information so that the log records from each component can be correlated, building up a complete picture of that transaction.

Because request metrics tracks individual transactions, using it imposes some performance implications on the system. However, this function can be mitigated by the use of the request filtering capabilities.

For example, tools can inject synthetic transactions. Request metrics can then track the response time within the WebSphere Application Server environment for those transactions. A synthetic transaction is one that is injected into the system by administrators to take a proactive approach to testing the performance of the system.

"Performance Monitoring Infrastructure (PMI) provides information about average system resource usage statistics, with no correlation between the data across different WebSphere® Application Server components. For example, PMI provides information about average thread pool usage. Request metrics provides data about each individual transaction, correlating this information across the various WebSphere Application Server components to provide an end-to-end picture of the transaction" ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cprf_positioningrm.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cprf_positioningrm.html)).

Enabling Request Metrics:
[https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_requestmetrics.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_requestmetrics.html) and
[https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_reenable.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_reenable.html) and
[https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/uprf_requestmetrics.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/uprf_requestmetrics.html) and

Description of ARM data in SystemOut.log:

After ARM is enabled, to get data in the web server plugin, you must regenerate the configuration file:

**Enabling Request Metrics**

The overhead of "Standard Logs" may be in the tens of percent or more, mostly due to the additional volume of logging. Consider using HPEL if available to reduce this.

- WebSphere Administrative Console > Monitoring and Tuning > Request Metrics
- Ensure "Prepare Servers for Request metrics collection" is checked (by default, it is).
- Under "Components to be instrumented," either select "All" or select "Custom," and multi-select the components; for example, "Servlet," "Servlet Filter," and "WebServices"
- Under "Trace level," select "Performance_Debug," unless you also need to see Servlet Filters, in which case select "Debug"
- Under "Request Metrics Destination," check "Standard Logs"
- Click "OK," save and synchronize. If "Prepare Servers for Request metrics collection" was already checked (the default), then the application server(s) do not need to be restarted.
- The output will go to SystemOut.log and it may be significant. Ensure that enough SystemOut historical files and sizes are configured: [http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/utrbJvmlogs.html?cp=SSAW57_8.5.5%2F3-18-6-479&lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/utrbJvmlogs.html?cp=SSAW57_8.5.5%2F3-18-6-479&lang=en)
  - For example this will write up to 1GB:
    - File Size > Maximum Size = 250MB
    - Maximum Number of Historical Log Files = 4

Here is example output (the elapsed portion is in milliseconds):

```
[11/7/13 15:11:45:178 PST] 00000008a PmiRmArmWrapp I
```
Note that request metrics is enabled at a cell level. Therefore, once the setting changes are saved and synchronized, all servers will immediately start logging request and this can impact performance on all of them. You can disable this on some servers by appending the diagnostic string com.ibm.ws.pmi.*=none before applying the setting changes.

Request Metrics Analyzer Next

The following GUI tool is a very nice way to explore request metrics logs:
https://github.com/skliche/request-metrics-analyzer-next

Request Metrics Filters

Request Metrics has a dramatic performance overhead when tracking every request, but it has various filters that only print data for requests that match the filters. One technique to use this in production is to add a filter for a particular IP address. When a problem occurs, use this client computer to make requests and that way you will see how the various components are responding for just those requests.

- Click on "Filters"
  - Click on "SOURCE_IP"
    - Check "Enable"
    - Click OK
  - Click on "Filter Values"
    - Click "New"
      - Value=$IP_ADDRESS
      - Check "Enable filter"
      - Click OK

If you are not seeing something, first confirm all the above are checked (sometimes settings are lost because of not clicking OK on the proper screens). Next, confirm you're using the right IP address. You can turn on NCSA access logging in WAS to see what the IP address is of the incoming user (see below).

Logging and Tracing

Unless you are consuming the JMX notifications for log events, disable them to improve performance (use system property -Dcom.ibm.ejs.ras.disablerasnotifications=true). This will avoid creating a JMX notification for every log entry.

Starting in WAS 8, the IBM service log (activity.log) is disabled by default. Before WAS 8, it is recommended to disable the activity.log.
Trace Overhead

The overhead of WAS diagnostic trace is proportional to the breadth of the trace specification and the number of concurrent threads (e.g. requests) driving said tracers. The overhead is inversely proportional to the filesystem speed, available system capacity (e.g. CPU & caches), number of CPU core threads, and available physical memory. It's very difficult to estimate the overhead of a trace specification, even for those that are commonly used, because just one of these variables may have a significant effect. For example, the broad WAS security trace which enables all security tracers may have very different overhead depending on which security features are configured. Therefore, a customer should run a baseline performance test that's representative of production traffic in a test environment, and then run the same test with the desired trace enabled, and calculate the overhead.

In one DayTrader benchmark, the diagnostic trace ejbcontainer=fine, which is a detailed trace of EJB activity, reduced throughput by 75%. Starting with WAS 8, the optional High Performance Extensible Logging (HPEL) diagnostic trace alternative (with TextLog disabled) reduced that same benchmark overhead by 50%. With both WAS diagnostic trace systems, if log statement JMX notifications are not needed, -Dcom.ibm.ejs.ras.disablerasnotifications=true should be used.

Here are some ideas to improve the trace experience:

1. On WAS >=8, switch to HPEL with the TextLog disabled (for convenience, the TextLog may be enabled for only a slightly penalty as it doesn't contain traces).
2. Tune the speed of the filesystem where the trace is written.
3. Consider using operating system RAMdisks to dedicate RAM to a virtual filesystem and write the traces to that mount.
4. If possible, use the generic JVM argument -Dcom.ibm.ejs.ras.disablerasnotifications=true
5. If the problem can be reproduced with a single user, isolate a production server from production traffic, enable all the full required traces, and use some mechanism to only allow one problematic user onto that server (e.g. direct IPs, ODR routing rules, etc.).
6. Disable the IBM service log (activity.log). On WAS 8, and later versions, it is disabled by default.
7. If trace is being written to a networked filesystem, write to a local filesystem instead (or RAMdisk).
8. Ask IBM development for a reduced trace string or diagnostic patch with very specific trace points.

Controlling Trace Levels

The diagnostic trace level defaults to *=info. The level specification is a colon-delimited list of the form name=level and it may be changed dynamically at runtime: http://www-01.ibm.com/support/docview.wss?uid=swg21254706

Depending on the trace specification and application activity, the volume of trace written may be very high. It is often recommended to update the trace log rotation to something like File > Maximum Size = 250MB and Maximum Number of Historical Log Files = 4 (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/utrb_traceservice.html?cp=SSAW57_8.5.5%2F3-18-6-295&lang=en).

Print the current trace level using `wsadmin -lang jython`:

```python
AdminControl.getAttribute(AdminControl.completeObjectName("type=TraceService,
```
Dynamically update trace level using `wsadmin -lang jython`:

```python
AdminControl.setAttribute(AdminControl.completeObjectName("type=TraceService, process=server1,*"), "traceSpecification", "*=all")
```

In WAS >= 7.0.0.37, 8.0.0.10, and 8.5.5.5, a new setTraceSpecification method has been added which returns the finally applied string (for verification or typos and optimizations):

```python
AdminControl.invoke(AdminControl.completeObjectName("type=TraceService,process=server1,*"), "setTraceSpecification", "*=all:*=info")
```

The diagnostic trace level may also be used to control java.util.logging.Logger (JUL) thresholds. Here is an example servlet with a JUL:


If the WAS diagnostic trace level is set to *=info: com.ibm.simpleweb.SimpleWebServlet=all, then trace.log will show matching JUL statements:

```
[10/6/14 12:45:15:158 PDT] 0000009f SimpleWebServ >
com.ibm.simpleweb.SimpleWebServlet.service ENTRY
[10/6/14 12:45:15:159 PDT] 0000009f SimpleWebServ <
com.ibm.simpleweb.SimpleWebServlet.service RETURN
```

However, you will receive the following warning when using such a specification in the administrative console. This warning may be disregarded.

```
The configured trace state included the following specifications that do not match any loggers currently registered in the server: "com.ibm.simpleweb.SimpleWebServlet=all"
```

### High Performance Extensible Logging (HPEL)

Starting in WAS 8, use High Performance Extensible Logging. HPEL will greatly improve your log/trace performance, and will make it easy to separate your logs per application, filter through logs and trace, and will enable developers to extend log and trace records with custom context (http://www.ibm.com/developerworks/websphere/techjournal/1208_bourne/1208_bourne.html):


If possible, disable the HPEL text log to further improve performance. The text log content is redundant: the same info is stored in the HPEL log and trace binary repositories. Examples of how to change some HPEL settings via Jython are at [http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/ttpb_confHPELwsadmin.html?cp=SSAW57_8.5.5%2F3-10-22-7&lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/ttpb_confHPELwsadmin.html?cp=SSAW57_8.5.5%2F3-10-22-7&lang=en). The logViewer command
offers most of the features that administrators expect for log viewing, such as tailing (see below).

Benchmarks using DayTrader with diagnostic trace ejbcontainer=fine (plus text log disabled and -Dcom.ibm.ejs.ras.disablerasnotifications=true) show that HPEL is only about 50% worse than no trace, whereas traditional diagnostic tracing is about 75% worse than no trace (i.e. HPEL has 50% higher throughput than traditional diagnostic trace).

logViewer

The logViewer tool is used to read binary HPEL logs. There are various options, including a -monitor [seconds] option to dynamically tail logs (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rtrb_logviewer.html?lang=en):

$ logViewer -monitor

Cross Component Trace (XCT)

XCT is available starting in WAS 8.5. XCT adds a unique request identifier to log and trace entries. XCT is similar to request metrics in many ways, but it is more deeply ingrained into the flow. XCT requires that High Performance Extensible Logging (HPEL) is enabled instead of classic logging, and you also have to enable XCT itself.

The IBM Support Assistant has a new tool called the XCT Log Viewer which visualizes the BEGIN/END records (not requestID information – in fact, it can't even take advanced format logs as input). The tool can load multiple files and correlate the events in a tree view.

There are four XCT modes: Disabled, Enabled, Enabled+XCT Records, Enabled+XCT Records+Data Snapshots. The simple Enabled mode adds a unique request ID to every applicable log and trace record. You can dump this data using the HPEL logViewer with the "-format advanced" argument. For example, I've got an application that causes a transaction timeout. Traditionally, all you would get is a WTRN0124I message with the last thread stack and a WTRN0041I message noting the timeout. I enabled the minimal tracing of getting WAS response times and then ran logViewer -format advanced:

[7/10/12 9:11:45:121 PDT] 00000099 I UOW= source=com.ibm.websphere.XCT org=null prod=null component=null thread=[WebContainer : 2] requestID=[AABHT9d/5yd-AAAAAAAAAAB] BEGIN AABHT9d/5yd-AAAAAAAAAAB 00000000000-cccccccccc2 HTTPCF(InboundRequest /TransactionTest/Test RemoteAddress(0:0:0:0:0:0:0:1) RequestContext(2072483128))
[7/10/12 9:13:45:125 PDT] 0000007e I UOW= source=com.ibm.ws.tx.jta.TimeoutManager org=IBM prod=WebSphere component=Application Server thread=[Non-deferrable Alarm : 1] WTRN0124I: When the timeout occurred the thread with which the transaction is, or was most recently, associated was Thread[WebContainer : 2,5,main]. The stack trace of this thread when the timeout occurred was: ...

First Failure Data Capture (FFDC)

Since 7.0.0.19, after an FFDC exception is thrown, the algorithm is here: http://www-01.ibm.com/support/docview.wss?uid=swg1PM39875
"...for the FFDC summary file to be updated for a given incident...
1. When there have been more than 10 incidents and at least a minute has
   passed after the last time the summary table was updated.
2. It has been more than 5 minutes since the last time the summary table was
   updated."

When this happens, the same file name is used - ${server}_exception.log - but the file is simply
truncated and rewritten.

The _exception.log file is only rotated on JVM startup: [http://www-01.ibm.com/support/docview.wss?uid=swg1PK86345](http://www-01.ibm.com/support/docview.wss?uid=swg1PK86345)

The FFDC1003I message is only printed the first time each "type" of an FFDC exception is thrown.
After that, only the summary _exception.log file is updated. This can be configured differently but it
would create a lot more FFDC log files.

Example _exception.log:

```
<table>
<thead>
<tr>
<th>Index</th>
<th>Count</th>
<th>Time of first Occurrence</th>
<th>Time of last Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Index  Counter</td>
<td>Exception SourceId</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count  Time of first Occurrence</td>
<td>Time of last Occurrence</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>--------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>10/20/14 10:54:32:479 PDT</td>
<td>10/20/14 11:05:32:584 PDT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>java.io.IOException</td>
<td>com.ibm.ws.management.discovery.DiscoveryService.sendQuery 189 ...txt</td>
</tr>
<tr>
<td></td>
<td></td>
<td>org.omg.CORBA.INV_OBJREF</td>
<td>com.ibm.ws.giop.message.GIOPRequestMessage.populate 192 ...txt</td>
</tr>
</tbody>
</table>
```

**Transaction Log**

The Transaction log directory can be set in the administrative console by navigating to Servers =>
Application Servers => server_name => Container Services => Transaction Service.

When an application that runs on the application server accesses more than one resource,
the application server stores transaction information in the product directory so that it can
coordinate and manage the distributed transaction correctly. When there is a higher
transaction load, storing persistent information in this way can slow the performance of the
application server because it depends on the operating system and the underlying storage
systems. To achieve better performance, designate a new directory for the log files on a
separate, physically larger, storage system.


The transaction log is most commonly stored either in a shared filesystem or in a database. In general,
internal benchmarks show that using a database is approximately 10% slower, but the time spent
processing the transaction log is usually a small proportion of the total transaction time, so this
difference is often imperceptible.
Networked Filesystem (NFS)

CPU Starvation Detected Warning (HMGR0152W)

Current thread scheduling delay is 7 seconds.

The HMGR0152W starvation detection warning works by looping, noting time X, calling
java/lang/Thread.sleep(Y=IBM_CS_THREAD_SCHED_DETECT_PERIOD, default 30 seconds),
noting time Z upon return, and then reporting Z-Y-X as the scheduling delay if it is over the threshold
IBM_CS_THREAD_SCHED_DETECT_ERROR (default 5 seconds).

For example, by default, a report of a 7 second scheduling delay means that a thread called
Thread.sleep(30), but returned 37 seconds later, 2 seconds more than the threshold.

If this message appears frequently, or if this message occurs at about the same time as significant
performance slowdowns or timeouts, you may want to investigate further. This message will disappear
when the thread scheduling delay has been corrected. Perform resource analysis to determine the
proper course of action. Common items to review:

- The most common cause of this is a long, stop-the-world garbage collection cycle, because Java
  threads, including the timer that prints this warning, cannot be dispatched during this cycle.
  Review verbose:gc or a monitoring tool for garbage collections immediately preceding this
  warning that take longer than IBM_CS_THREAD_SCHED_DETECT_ERROR.
- Review operating system statistics immediately preceding the warning such as high processor
  utilization, processor run queues greater than available processors, low memory and paging
  activity, virtualization steal times, etc. Operating system statistics are often gathered at intervals
  such as 60 or 300 seconds. If this interval is greater than
  IBM_CS_THREAD_SCHED_DETECT_ERROR, then the relevant symptoms may be
  averaged out of the operating system numbers. In this case, reduce the operating system
  statistics gathering interval to less than or equal to the
  IBM_CS_THREAD_SCHED_DETECT_ERROR.

Thread Pools

Thread pools and their corresponding threads control all execution of the application. The more threads
you have, the more requests you can be servicing at once. However, the more threads you have the
more they are competing for shared resources such as CPUs and the slower the overall response time
may become as these shared resources are contended or exhausted. If you are not reaching a target CPU
percentage usage, you can increase the three pool sizes, but this will probably require more memory
and should be sized properly. If there is a bottleneck other than the CPUs, then CPU usage will stop
increasing.

You can think of thread pools as queuing mechanisms to throttle how many concurrent requests you
will have running at any one time in your application.

The most commonly used (and tuned) thread pools within the application server are:
1. WebContainer: Used when requests come in over HTTP.
2. ORB: Used when remote requests come in over RMI/IIOP for an enterprise bean from an EJB application client, remote EJB interface, or another application server.
3. Messaging thread pools (see the messaging chapter)

Understand which thread pools your application uses and size all of them appropriately based on utilization you see in tuning exercises through thread dumps or PMI/TPV.

If the application server ends up being stalled 1/2 of the time it is working on an individual request (likely due to waiting for a database query to start returning data), then you want to run with 2X the number of threads than cores being pinned. Similarly if it's 25%, then 4X, etc.

Use TPV or the IBM Thread and Monitor Dump Analyzer (https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c) to analyze thread pools.

Thread pools need to be sized with the total number of hardware processor cores in mind.

- If sharing a hardware system with other WAS instances, thread pools have to be tuned with that in mind.
- You need to more than likely cut back on the number of threads active in the system to ensure good performance for all applications due to context switching at OS layer for each thread in the system
- Sizing or restricting the max number of threads an application can have, will help prevent rouge applications from impacting others.

The ActiveCount statistic on a thread pool in WebSphere is defined as "the number of concurrently active threads" managed by that thread pool. This metric is particularly useful on the WebContainer thread pool because it gives an indication of the number of HTTP requests processed concurrently.

Note: The concurrent thread pool usage (PMI ActiveCount) may not necessarily be the concurrently "active" users hitting the application server. This is not due just to human think times and keepalive between requests, but also because of asynchronous I/O where active connections may not be actively using a thread until I/O activity completes (non-blocking I/O). Therefore, it is incorrect to extrapolate incoming concurrent activity from snapshots of thread pool usage.

If this metric approaches its maximum (which is determined by the maximum pool size), then you know that either the pool is simply too small or that there is a bottleneck that blocks the processing of some of the requests.

- **Thread pool**- Parameters: Good practice is to use 5 threads per server CPU core for the default thread pool, and 10 threads per server CPU for the ORB and Web container thread pools. For a machine with up to 4 CPUs, the default settings are usually a good start for most applications. If the machine has multiple application server instances, then these sizes should be reduced accordingly. Conversely, there could be situations where the thread pool size might need to be increased to account for slow I/O or long running back-end connections. Ref: http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html
Hung Thread Detection

WAS hung thread detection may be more accurately called WAS long response time detection (default 10-13 minutes) and the "may be hung" warning may be more accurately read as "has been executing for more than the configured threshold."

WSVR0605W is the warning printed when WAS detects that a unit of work is taking longer than the WAS hung thread detection threshold. The default hung thread detection threshold is 10 minutes. Hang detection only monitors most WAS managed threads, such as the WebContainer thread pool. Any native threads, or threads spawned by an application are not monitored. In recent versions of WAS, the warning includes the stack which often points to the delay:

[11/16/09 12:41:03:296 PST] 00000020 ThreadMonitor W WSVR0605W: Thread "WebContainer : 0" (00000021) has been active for 655546 milliseconds and may be hung.
There is/are 1 thread(s) in total in the server that may be hung.
  at java.lang.Thread.sleep(Native Method)
  at java.lang.Thread.sleep(Thread.java:851)
  at com.ibm.Sleep.doSleep(Sleep.java:55)
  at com.ibm.Sleep.service(Sleep.java:35)
  at javax.servlet.http.HttpServlet.service(HttpServlet.java:831)...

WAS will check threads every `com.ibm.websphere.threadmonitor.interval` seconds (default 180) and any threads dispatched more than `com.ibm.websphere.threadmonitor.threshold` seconds (default 600) will be dumped (unless the false alarm threshold has been hit). Therefore, any thread dispatched between `com.ibm.websphere.threadmonitor.threshold` seconds and
`com.ibm.websphere.threadmonitor.threshold` + `com.ibm.websphere.threadmonitor.interval` seconds will be marked.

The amount of time the thread has been active is approximate and is based on each containers' ability to accurately reflect a thread's waiting or running state; however, in general, it is the amount of milliseconds that a thread has been dispatched and doing "work" (i.e. started or reset to "non waiting" by a container) within a WAS managed thread pool.

The various hung thread detection properties may be changed and they are effective after a restart:

```
```

- `com.ibm.websphere.threadmonitor.interval`: The frequency (in seconds) at which managed threads in the selected application server will be interrogated. Default: 180 seconds (three minutes).
- `com.ibm.websphere.threadmonitor.threshold`: The length of time (in seconds) in which a thread can be active before it is considered hung. Any thread that is detected as active for longer than this length of time is reported as hung. Default: The default value is 600 seconds (ten minutes).
- `com.ibm.websphere.threadmonitor.dump.java`: Set to true to cause a javacore to be created when a hung thread is detected and a WSVR0605W message is printed. The threads section of the javacore can be analyzed to determine what the reported thread and other related threads are doing. Default: False. Note: On z/OS, dumpThreads also creates a heapdump and TDUMP by default. This may be controlled with wsadmin_dumpthreads_enable_heapdump and wsadmin_dumpthreads_enable_javatdump: [http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/urun_rproperty_custproperties.html?lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/urun_rproperty_custproperties.html?lang=en)
- Starting with WAS 8.0.0.10 and 8.5.5.5 ([http://www-01.ibm.com/support/docview.wss?uid=swg1PI27232](http://www-01.ibm.com/support/docview.wss?uid=swg1PI27232)), `com.ibm.websphere.threadmonitor.dump.java.track`: Set to an integer value in the range 2 through the value of `com.ibm.websphere.threadmonitor.dump.java` to cause the dumpThreads function to be run over subsequent monitor intervals in which a thread remains hung. The integer value indicates the maximum number of times dumpThreads will be run to track a hung thread.

For IBM JVMs, you can also produce dumps on a hung thread warning ([https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/trace_options_trigger.html](https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/trace_options_trigger.html)):

- `Xtrace:trigger=method{com/ibm/ws/runtime/component/ThreadMonitorImpl$RasListerner.threadIsHung,sysdump,,,1}`

On WAS &gt;= 8.5:

- `Xtrace:trigger=method{com/ibm/ws/runtime/component/ThreadMonitorImpl.threadIsHung,sysdump,,,1}`

In this example, the maximum number of system dumps to produce for this trigger is 1. Enabling certain -Xtrace options may affect the performance of the entire JVM (see the -Xtrace section in the IBM Java chapter).
Thread Pool Statistics

Starting with WAS 7.0.0.31, 8.0.0.8, and 8.5.5.2, thread pool statistics may be written periodically to SystemOut.log or trace.log. This information may be written to SystemOut.log by enabling the diagnostic trace Runtime.ThreadMonitorHeartbeat=detail or to trace.log by enabling the diagnostic trace Runtime.ThreadMonitorHeartbeat=debug. Example output:

```
[1/12/15 19:38:15:208 GMT] 000000d4 ThreadMonitor A
UsageInfo[ThreadPool:hung/active/size/max]={[SIBFAPThreadPool:0/2/4/50,
TCPChannel.DCS:0/3/18/20,
server.startup:0/0/1/3,
WebContainer:0/3/4/12,
SIBJMSRAThreadPool:0/0/10/41,
ProcessDiscovery:0/0/1/2,
Default:0/2/7/20,
ORB.thread.pool:0/0/10/77,
HAManager.thread.pool:0/0/2/2}
```

When the diagnostic trace is enabled, this output is written every com.ibm.websphere.threadmonitor.interval seconds. Only thread pools that have at least one worker thread (whether active or idle) will be reported.

BoundedBuffer


The thread pool request buffer is essentially a backlog in front of the thread pool. If the thread pool is at its maximum size and all of the threads are dispatched, then work will queue in the requestBuffer. The maximum size of the requestBuffer is equal to the thread pool maximum size; however, if the unit of work is executed on the thread pool with a blocking mode of EXPAND_WHEN_QUEUE_IS_FULL_ERROR_AT_LIMIT or EXPAND_WHEN_QUEUE_IS_FULL_WAIT_AT_LIMIT, then the maximum size is ThreadPoolMaxSize * 10. When the requestBuffer fills up, then WSVR0629I is issued (although only the first time this happens per JVM run per thread pool). When the requestBuffer is full, work will either wait or throw a ThreadPoolQueueIsFullException, depending on how the unit of work is executed.

Java Database Connectivity (JDBC)

Database Connection Pools

Database connection pools are highly contended in heavily multi-threaded applications. Ensuring available connections in the pool leads to superior performance. Monitor PMI metrics to watch the number of threads waiting on connections from the pool as well as the average wait time.

- If threads are waiting, consider increasing the number of pooled connections in conjunction
with your database administrator (DBA), decreasing the number of active threads in the system, or investigating the usage of database connections by the application.

- In some cases, a one-to-one mapping between DB connections and threads may be ideal.
- Always use the latest database driver for the database you are running as performance optimizations between versions are often significant.

Connection pool settings:

If there is a firewall between WAS and the database and the firewall has a timeout on inactive sockets, then consider modifying the "Unused timeout" and "Reap time" settings. In the worst case (assuming normal scheduling conditions), a WAS connection may be unused up to "Unused timeout" + "Reap time," so make sure the sum of these values is less than the firewall timeout.

The DataSource MBean may be used to query connection pool usage using `wsadmin -lang jython`. In the following example, three connections are in use and two connections are free:

```
wsadmin> print
AdminControl.invoke(AdminControl.queryNames("*:type=DataSource,process=server1,name=TradeDataSource,*"), "showPoolContents")

PoolManager name:jdbc/TradeDataSource
PoolManager object:-522043580
Total number of connections: 5 (max/min 5/5, reap/unused/aged 180/1800/0,
connectiontimeout/purge 180/EntirePool)
 (testConnection/inteval false/0, stuck
timer/time/threshold 0/0/0, surge time/connections 0/-1)
 (pool paused false, prePopulate alternate
false, resourceFailBackEnabled true, isAlternateResourceEnabled false,
disableDatasourceFailoverAlarm false,
startFailBack false)
(isPartialResourceAdapterFailoverSupportEnabled false,
isAlteranteResourcePoolManager false, resourceAvailabilityTestRetryInterval 10, currentInusePool null, currentMode 100,
alternate jndiName null)
Shared Connection information (shared partitions 200)
com.ibm.ws.tx.jta.TransactionImpl@a47615d6#tid=349227028  MCWrapper id 767a05e9  Managed connection WSRdbManagedConnectionImpl@37f2f2c5
  State:STATE_TRAN_WRAPPER_INUSE Connections being held 1 Used with
    transaction com.ibm.ws.tx.jta.TransactionImpl@a47615d6#tid=349227028
    com.ibm.ws.tx.jta.TransactionImpl@9ea5a8b5#tid=349227084  MCWrapper id 3f4efeef9  Managed connection WSRdbManagedConnectionImpl@689ac78c
    State:STATE_TRAN_WRAPPER_INUSE Connections being held 1 Used with
      transaction com.ibm.ws.tx.jta.TransactionImpl@9ea5a8b5#tid=349227084
      com.ibm.ws.tx.jta.TransactionImpl@4850aa55#tid=349227060  MCWrapper id 716535f  Managed connection WSRdbManagedConnectionImpl@7424ebb6
      State:STATE_TRAN_WRAPPER_INUSE Connections being held 1 Used with
        transaction com.ibm.ws.tx.jta.TransactionImpl@4850aa55#tid=349227060
        Total number of connection in shared pool: 3
      Free Connection information (free distribution table/partitions 5/1)
       (0)(0)MCWrapper id 863b69f0  Managed connection
       WSRdbManagedConnectionImpl@41038936 State:STATE_ACTIVE_FREE
       (0)(0)MCWrapper id 94ff7816  Managed connection
```
Total number of connection in free pool: 2
UnShared Connection information
No unshared connections

Connection Leak Logic Information: ...

All data source connection pool statistics may be displayed with showAllPoolContents:

```java
wsadmin>print
AdminControl.invoke(AdminControl.queryNames("*:type=DataSource,process=server1,name=TradeDataSource,*"), "showAllPoolContents")
```

Free connections in a data source connection pool may be purged manually:

```java
wsadmin>AdminControl.invoke(AdminControl.queryNames("*:type=DataSource,process=server1,name=TradeDataSource,*"), "purgePoolContents", "immediate")
```

In order to successfully tune the connection pool, you need to know two pieces of information:

1. The requests per second that occur during a peak
2. How long the database takes to respond to each type of operation, SELECT, INSERT, UPDATE, and so on.

Maximum connections setting:

- Double the number of the Maximum connections parameter then slowly back it down
- Better performance is generally achieved if this value is set lower than the value for the maximum size of the Web container thread pool

If a ConnectionWaitTimeoutException is found in the WebSphere logs:

- Obtain the average database operations duration for the application
- Start with a value that is 5 seconds longer than this average
- Gradually increase it until problem is resolved or setting is at the highest value that the client/SLAs will tolerate.
- Before you increase the pool size, consult the database administrator. Why? Because the DBA sets the maximum number of connections their database will accept. If the size of the connection pool increases then that will across all cluster members and can result in trying to establish more connections to the database than it will accept. That scenario results in a lot of strange failures that will take some time to troubleshoot to get to the root cause.
- Ensure that the database server is configured to handle the maximum pool size setting.
- In a clustered environment, there is the potential of simultaneously allocating Max connections form all servers simultaneously.

Connection pools are a shared, synchronized resource. They have been highly optimized but when there are a very large number of threads, lock synchronization may become a bottleneck. You may use the IBM Health Center tool or similar tool to measure the lock contention, and if it is high, then you may need to consider scaling out to more JVMs.
JDBC traces for DB2 and Oracle: http://www-01.ibm.com/support/docview.wss?uid=swg21654565

Statement cache

"The WebSphere Application Server data source optimizes the processing of prepared statements and callable statements by caching those statements that are not being used in an active connection. Both statement types help reduce overhead for transactions with backend data.

A prepared statement is a precompiled SQL statement that is stored in a PreparedStatement object. Application Server uses this object to run the SQL statement multiple times, as required by your application run time, with values that are determined by the run time. A callable statement is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the CallableStatement object. Application Server uses this object to run a stored procedure multiple times, as required by your application run time, with values that are determined by the run time.

In general, the more statements your application has, the larger the cache should be. Be aware, however, that specifying a larger statement cache size than needed wastes application memory and does not improve performance.

Determine the value for your cache size by adding the number of uniquely prepared statements and callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server.

Default: For most databases the default is 10. Zero means there is no cache statement." (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rdat_datobjtune.html)

The statement cache size specifies the number of statements that can be cached per connection. Caching prepared statements improves overall response times because an application can reuse a PreparedStatement on a connection if it exists in that connection's cache, bypassing the need to create a new PreparedStatement. However, to make effective use of this cache the application has to be properly written to use parameterized SQL statements using the ? (question mark) notation instead of dynamically building strings with the parameters already substituted as each unique statement will make the cache useless.


Ideally the PreparedStatementDiscardCount should be zero; however, if memory utilization considerations are important, then having a slow incrementing count is not necessarily a bad thing. Recommendations are made in several WebSphere Application Server documents on the value for the prepared statement cache. They all recommend estimating the number of unique SQL statements an application prepares and using this number to set the number of
prepared statements to be cached for each connection.

These formulas work well when the number of unique prepared statements and maximum connections are relatively small; however, these formulas do not take into account the possible memory consumption of the cached prepared statements, particularly when the total number of statements being cached is large. What is considered a small or large prepared statement cache depends on the database vendor in use.

Each prepared statement object consumes some amount of memory. The actual amount is variable, based on the database vendor in use, as well as the number and size of the parameter data for the statement. When prepared statement caches are configured to large values, it is possible to outgrow the amount of memory available to the cache, resulting in unexpected behavior. Depending on the type of JDBC driver, the memory consumption might be from the Java heap or from the JVM's native heap...

If you choose to decrease the size of your prepared statement cache, some cycling of the statement cache could occur, as the least recently used statements are closed to make room for more recently used statements. It can be worthwhile to analyze the usage pattern of the prepared statements in your application. If some prepared statements are executed infrequently, the penalty in consumed resources might outweigh the advantage of the caching mechanism. These infrequently-used statements might be better suited to the java.sql.Statement interface, rather than the java.sql.PreparedStatement interface. Statement objects are not cached by the Application Server and will not consume memory beyond the scope in which they are used.

http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg21108496

Shareable versus Unshareable Connections

Database connections marked shareable are not returned to the connection pool when they are closed. Instead, they are reserved for reuse by subsequent requests for a connection within the same transaction containment context. For example, if a thread within a servlet uses the normal get-use-close pattern on a database connection more than once, the second time, the same connection is immediately returned since it was reserved from the pool.

The Java Enterprise Edition specification defines shareable as the default configuration unless otherwise specified:

Sharing connections typically results in efficient use of resources and better performance. An application can indicate the ability to share its various resource references, or connections, in its deployment descriptor. A connection can be marked either as shareable or unshareable. The default is shareable.

With all that said, there are some cases where unshareable connections perform better, so you should consider trying unshareable. Note that this may expose connection leaks or other problems. You can set globalConnectionTypeOverride=unshared to disable shareable connections:
https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tdat_co
Scenarios where unshareable connections may be preferable:

- The time it takes for the application to service an HTTP request takes a long time.
- The application typically does not open/close more than one connection to service an HTTP request.
- The application rarely uses a transaction other than auto-commit with the database.

Scenarios where shareable connections may be preferable:

- The time it takes for the application to service an HTTP request is very quick.
- The application will frequently open/close a connection to the database.
- The application makes heavy use of transactions to the database.
- Some EJB container transactions require shareable connections.

As with any setting within the application server it is imperative to perform load testing and seeing which connection setting works better with the application.

**More JDBC Connections than Threads**

Applications that open more than one JDBC connection simultaneously in the same thread before closing the previous connections are identified by seeing more connections in the JDBC connection pool than threads in the thread pool. This can potentially result in an application deadlock if there are not enough connections in the connection pool. To correct this the application developers have to fix the code to close a JDBC connection before acquiring another connection.

**DB2 JDBC Driver**

On HP-UX, preallocate the DB2 trace segment and ensure the database is created with the UTF-8 code set:


**Oracle JDBC Driver**

Enabling Oracle JDBC driver trace: Oracle ships several JAR files for each version of the JDBC drivers. The optimized JAR files do not contain any logging code and, therefore, do not generate any log output when used. To get log output, you must use the debug JAR files, which are indicated with a "/g" in the file name, like ojdbc5_g.jar or ojdbc6_g.jar.

- Set this diagnostic trace:
  *=info:WAS.j2c=all:RRA=all:WAS.database=all:Transaction=all:com.ibm.ws.oracle.logwriter= all
- Activate the Debug Library by creating the custom property: -Doracle.jdbc.Trace=true

The JVM must be restarted after the changes have been made to use the debug JAR. The Oracle trace points all come from the 'logwriter' component.

In all releases of WebSphere Application Server, including V6.1, V7.0, V8.0, and V8.5, high memory usage, java.lang.OutOfMemoryErrors, slow performance, and a large volume of garbage collection cycles may occur when the Oracle JDBC driver is used to connect to Oracle databases. This is due to
the memory management of the Oracle JDBC driver, and does not affect other JDBC drivers or databases.

In a heap dump, it can be seen that the Oracle JDBC driver stores a large amount of data in Connection and PreparedStatement objects. For example oracle.jdbc.driver.T4CConnection, oracle.jdbc.driver.PhysicalConnection$BufferCacheStore, oracle.jdbc.driver.BufferCache, and oracle.jdbc.driver.T4CPreparedStatement.

The problem is caused by the way that the Oracle JDBC driver manages memory. For full details, refer to the Oracle white paper, Oracle JDBC Memory Management

Some of the most relevant statements include:

"The Oracle JDBC drivers can use large amounts of memory. This is a conscious design choice, to trade off large memory use for improved performance. For the most part and for most users this has proved to be a good choice. Some users have experienced problems with the amount of memory the JDBC drivers use.

[...] Some users, mostly those with very large scale applications, have seen performance problems due to large heap size, garbage collector thrashing, and even OutOfMemoryExceptions. In subsequent releases the development team has worked to address those issues by improving the way the drivers use memory and by providing users with additional control to address specific problems.

[...] the size of the buffers depends not on the actual size of the row data returned by the query, but on the maximum size possible for the row data. After the SQL is parsed, the type of every column is known and from that information the driver can compute the maximum amount of memory required to store each column. The driver also has the fetchSize, the number of rows to retrieve on each fetch. With the size of each column and the number of rows, the driver can compute the absolute maximum size of the data returned in a single fetch. That is the size of the buffers.

[...] In the worst case, consider a query that returns 255 VARCHAR2(4000) columns. Each column takes 8k bytes per row. Times 255 columns is 2040K bytes or 2MB per row. If the fetchSize is set to 1000 rows, then the driver will try to allocate a 2GB char[]. This would be bad... The primary tool for controlling memory use is the fetchSize.

[...] Although Java memory management is quite good, allocating large buffers is expensive. It is not the actual malloc cost. That is very fast. Instead the problem is the Java language requirement that all such buffers be zero filled. So not only must a large buffer be malloc'ed, it must also be zero filled. Zero filling requires touching every byte of the allocated buffer. Modern processors with their multilevel data caches do ok with small buffers. Zero filling a large buffer overruns the processor data caches and runs at memory speed, substantially less than the maximum speed of the processor. Performance testing has repeatedly shown that allocating buffers is a huge performance drag on the drivers. This has led to a struggle to balance the cost of allocating buffers with the memory footprint required to save buffers for reuse.

[...] The 11.1.0.7.0 drivers introduce a connection property to address the large buffer problem. This
property bounds the maximum size of buffer that will be saved in the buffer cache... The connection property is -Doracle.jdbc.maxCachedBufferSize=N ... e.g. 100000. The default is Integer.MAX_VALUE. This is the maximum size for a buffer which will be stored in the internal buffer cache... If you need to set maxCachedBufferSize, start by estimating the buffer sizes for the SQL queries that require the largest buffers. In the process you may find that by tuning the fetch size for these queries you can achieve the desired performance. Considering the frequency of execution and the size of the buffers, pick a size such that most statements can use cached buffers, but still small enough so that the Java runtime can support the number of buffers needed in order to minimize the frequency with which new buffers have to be allocated.

[...] In 11.2 the value of maxCachedBufferSize is interpreted as the log base 2 of the maximum buffer size. For example if maxCachedBufferSize is set to 20 the max size buffer that is cached is 2^20 = 1048576. For backwards compatibility, values larger than 30 are interpreted as the actual size rather than log2 of the size, but using powers of 2 is recommended... It is usually the case that setting maxCachedBufferSize to a reasonable value has no impact. If you need to set maxCachedBufferSize, start with 18. If you have to set the value to less than 16, you probably need more memory."

**Servlets**

The number of persistent requests per connection may cause a significant throughput improvement, particularly with SSL. In one benchmark, 100% (http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html).

Servers > Application servers > $SERVER > Web container settings > Web container transport chains > * > HTTP Inbound Channel > Select "Use persistent (keep-alive) connections" and "Unlimited persistent requests per connection"

Disable application class and JSP reload checking:

- Enterprise Applications > $APP > Class loading and update detection
  - Check "Override class reloading settings for Web and EJB modules"
  - Set "Polling interval for updated files" = 0
- Enterprise Applications > $APP > JSP and JSF options
  - Uncheck "JSP enable class reloading"

Save, Synchronize, and Restart

If more than 500 unique URLs are actively being used (each JavaServer Page is a unique URL), you should increase the size of the invocation cache: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tweb_url_cache.html

**WebContainer Thread Pool**

If system resources allow, it is recommended to set the minimum size of the WebContainer equal to the maximum size because some DirectByteBuffers are cached and kept in thread locals and these are lost if the threads recycle. See http://www-01.ibm.com/support/docview.wss?uid=swg1PK24910
NCSA Access Logs

The HTTP transport channel supports the NCSA access log format to print a line for every HTTP response with various details such as URL (similar to the IHS access log): http://www-01.ibm.com/support/docview.wss?uid=swg21661868.

The original version of WAS NCSA access log did not support custom log formats and did not include the time the response took to be returned. Recent versions support the accessLogFormat custom property which allows a custom format, including %D: The elapsed time of the request in microseconds (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rrun_chain_httpcustom.html). This can be used as a lightweight and simple alternative to Request Metrics. This is available starting in WAS 7.0.0.25 (http://www-01.ibm.com/support/docview.wss?uid=swg1PM68250) and WAS 8.0.0.2 (http://www-01.ibm.com/support/docview.wss?uid=swg1PM46717); however, note that there is also an APAR with that function that is required available starting in 7.0.0.29 (http://www-01.ibm.com/support/docview.wss?uid=swg1PM81277). You should also first make sure that APAR PM86708 is applied: http://www-01.ibm.com/support/docview.wss?uid=swg1PM86708

In the WAS Administrative Console:

1. Navigate to ${SERVER} > Web Container Settings > Web container transport chains > WCInbound* > HTTP inbound channel
   1. Note that WCInbound* means applying this whole procedure for each inbound WebContainer transport chain which is used to serve the traffic of interest.
2. Check "Enable logging"
3. Expand "NCSA Access logging"
   1. Check "Use chain-specific logging"
   2. Access log file path = ${SERVER_LOG_ROOT}/http_access.log
   3. Access log maximum size = 500
   4. Maximum Number of historical files = 2
   5. NCSA access log format = Common
4. Expand "Error logging"
   1. Check "Use chain-specific logging"
   2. Error log file path = ${SERVER_LOG_ROOT}/http_error.log
   3. Error log maximum size = 500
   4. Maximum Number of historical files = 2
5. Click Apply
6. Click "Custom properties"
7. Click New...
   1. Name = accessLogFormat
   2. Value = %h %i %u %t "%r" %s %b %D
   3. Click OK
8. Save, synchronize, and restart the JVM.

For example, with an accessLogFormat of %h %i %u %t "%r" %s %b %D, an access.log will be written in ${WAS}/profiles/${PROFILE}/logs/ with output such as the following. The last column is the response time of the request in microseconds (divide by 1000 for milliseconds):

```
127.0.0.1 - - [03/Sep/2014:17:32:33 -0700] "GET / HTTP/1.1" 200 5792 25603
```
The time printed is the time the request arrived, so it is likely that the timestamps will not be in order.

Starting with WAS 8.5.5.5, \%{X}W may be used to print the XCT Context ID, if available:
http://www-01.ibm.com/support/docview.wss?uid=swg1PI29618

The following POSIX command may be used to review the top 5 slowest requests in the access.log:

```bash
$ awk '{print $NF,$0}' access.log | sort -nr | cut -f2- -d' ' | head -5
```

Clone the problemdetermination git repository and run httpchannel.sh (requires Perl and gnuplot) in the same directory as the access.log file:

```bash
$ git clone https://github.com/kgibm/problemdetermination
$ problemdetermination/scripts/was/httpchannel.sh access.log
```

Example:

![HTTP Response Time Milliseconds access.log (UTC)](chart.png)
WebContainer Channel Write Type

The design of WAS with the default configuration of channelwritetype=async is that WAS will buffer up to the size of each HTTP response in native DirectByteBuffer memory as it waits for asynchronous TCP writes to finish (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rweb_custom_props.html). This means that if WAS is serving a large volume of responses from Java servlets (including static files through the WAS FileServlet, servlet/JSP responses, etc.), and if the clients (or the network path leading to the clients) cannot keep up with the pace of network writes, then these DirectByteBuffers will consume the amount of pending writes in native memory. This can cause native OutOfMemoryErrors in 32-bit processes, or paging on 64-bit processes with insufficient physical memory. Even if the network and end-user do keep up, this behavior may simply create a large volume of DBBs that can build up in the tenured area. You may change channelwritetype to sync to avoid this behavior (http://www-01.ibm.com/support/docview.wss?uid=swg21317658) although servlet performance may suffer, particularly for end-users on WANs. A feature request has been opened to change the default behavior: https://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=53870

Note: With channelwritetype=async, you may see WCChannelLinks waiting to write to the client without any WebContainer thread processing a request. This is expected and is a possibility with asynchronous writing. In this case, what likely happened is that the servlet wrote all of its response to the HTTP channel and finished its use of the thread, and the HTTP channel will asynchronously write the buffered response to the client.

If you have a system dump, in the Memory Analyzer Tool, you can find DirectByteBuffers waiting to be written to the client in the writeQueue java.util.ArrayList under com.ibm.ws.webcontainer.channel.WCChannelLink. In a PHD heapdump, you won't know it is the writeQueue, but that field is the only ArrayList on that object so you know it is the writeQueue. Right click on the ArrayList and click Show Retained Set. Each com.ibm.ws.buffermgmt.impl.PooledWsByteBufferImpl references a DirectByteBuffer, so the number of these instances will correlate with the number of DirectByteBuffers. In a system dump, you can also check the "writing" field on the WCChannelLink to see if that link to the client is still in the process of writing the response.

If you have a system dump and a recent version of the IBM Extensions for Memory Analyzer, in the IBM Memory Analyzer Tool, you can determine the channelwritetype by clicking Open Query Browser > IBM Extensions > WebSphere Application Server > Web Container Analysis.

If you have a system dump, you can find the URL being processed (to review if it may be a large file, for example) and other information such as HTTP headers underneath the WCChannelLink request and response fields.

DirectByteBuffer Pools

The WAS WebContainer uses DirectByteBuffers (DBBs) to perform HTTP reads and writes. The use of DBBs is required for good performance. DBBs are used in both cases of channelwritetype=async and channelwritetype=sync. The way DBBs are used is that each WebContainer thread has a lazy-loaded, ThreadLocal pool of DBBs and there is a global pool of DBBs for all WebContainer threads. This is a major reason why it's good for performance to set the minimum size of the WebContainer thread pool to the maximum size because that minimizes the creation and destruction of these DBBs.
The size of the DBB used will depend on the size of the HTTP read or write. Each DBB pool is split into buckets with each bucket having DBBs of a certain fixed size. The default sizes of the DBBs are:
32, 1024, 8192, 16384, 24576, 32768, 49152, 65536

In other words, there is a bucket of DBBs that are each 32 bytes, and a bucket of DBBs that are each 1024 bytes, and so on.

The default sizes of each bucket for a WebContainer ThreadLocal DBB pool are:
30, 30, 30, 20, 20, 20, 10, 10

In other words, there can be up to 30 DBBs of size 32 in the first bucket, up to 30 DBBs of size 1024 in the second bucket, and so on.

The global DBB pool multiplies each of the bucket sizes by 10. In other words, there can be up to 300 DBBs of size 32, and so on.

Therefore, by default, the global pool will use up to ~28MB of DBB native memory, and each WebContainer ThreadLocal DBB pool will use up to ~3MB of DBB native memory.

To determine if the DBB sizes and/or DBB bucket sizes are insufficient, first, ensure that the WebContainer thread pool minimum = maximum, then configure DBB trace (this may have significant overhead, so be careful running in production) with -Xtrace:print=j9jcl.335-338,trigger=tpnid{j9jcl.335,jstacktrace},trigger=tpnid{j9jcl.338,jstacktrace}, run the JVM until the WebContainer thread pool reaches the maximum size, and run the workload until it reaches steady state. If after this point, the DBB trace is still showing allocations from com.ibm.ws.buffermgmt.impl.WsByteBufferPoolManagerImpl.allocateBufferDirect, then consider increasing the DBB and/or bucket sizes. Normally, we only change the bucket sizes (poolDepths) and leave the poolSizes as default.

Another inconclusive but often indirect symptom of DBB pool exhaustion is high global garbage collection pause times with high numbers of PhantomReferences being cleared. The native memory backing DirectByteBuffers is cleared using PhantomReferences, so once a DBB has no more strong references, it is put on a queue like a finalizer. DBBs tend to get tenured, so they can build up in the tenured region of a generational collector and this will hold on to native memory until the next full GC, or if MaxDirectMemorySize is hit, and a large number of queued DBBs may increase global GC pause times (in some implementations, because PhantomReference processing is single threaded).

To modify either the DBB sizes and/or the bucket sizes, edit server.xml (in a network deployment environment, edit in the deployment manager configuration and then synchronize the node(s)):

In the root process:Server element, add the attribute

```
```

Find the services element with the xml:type loggingservice.http:HTTPAccessLoggingService. After the matching </services> tag, override the DBB sizes and/or the bucket sizes. For example:

```xml
<services xml:type="wsbytebufferservice:WSByteBufferService" xml:id="WSBBS_1" enable="true">
  <properties xml:id="BuffSVC_4" name="poolSizes" value="32,1024,8192,16384,24576,32768,49152,65536"/>
  <properties xml:id="BuffSVC_5" name="poolDepths"
```
Restart the JVM.

**JSP Buffers**

The JSP body buffer needs to contain the evaluation of a JSP body tag. The buffer will grow to the size of the body of an action: "The buffer size of a BodyContent object is unbounded." ([http://docs.oracle.com/javaee/6/api/javax/servlet/jsp/tagext/BodyContent.html](http://docs.oracle.com/javaee/6/api/javax/servlet/jsp/tagext/BodyContent.html)). The property BodyContentBuffSize defines the initial size of each buffer (default 512 bytes) and it's doubled until all of the content is contained. If com.ibm.ws.jsp.limitBuffer=false (the default), the buffer will remain at its latest size for subsequent requests. If com.ibm.ws.jsp.limitBuffer=true, the buffer is reset to BodyContentBuffSize. If the total size of instances of org.apache.jasper.runtime.BodyContentImpl exceeds 5-10% of the maximum Java heap size, then it's recommended to either reduce the application's usage of large JSP body content and/or to set com.ibm.ws.jsp.limitBuffer=true ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rweb_custom_props.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rweb_custom_props.html)).

It's difficult to theoretically calculate an optimal default value for BodyContentBuffSize. If the size is too small, then there is potentially extra time spent growing the buffer. If the size is too large, then there is potentially extra time spent garbage collecting. This is a property used for all JSPs, but if there are multiple JSPs, they will have different characteristics. As with most performance tuning, the best approach is to test different options and find the optimal value using a binary search (ideally first in a test environment): Start with a value X1=512. Continue doubling as long as results improve. Once results are worse, halve the difference from the previous value (X2-X1)/2 and repeat the algorithm (double or halve the difference) until an optimal value is found. If you have a heapdump, use the Memory Analyzer Tool to calculate the retained set of the class org.apache.jasper.runtime.BodyContentImpl.

If you have a system dump (IBM Java) or HPROF heapdump (HotSpot), then use the following OQL queries in the Memory Analyzer Tool to check the settings of limitBuffer and BodyContentBuffSize:

```sql
SELECT x.limitBuffer FROM InstanceOf java.lang.Class x WHERE x.@displayName.contains("class org.apache.jasper.runtime.BodyContentImpl ")

true
```

```sql
SELECT x.bodyContentBufferSize FROM org.apache.jasper.runtime.JspFactoryImpl x

512
```

**HTTP gzip compression**

HTTP compression can be done either for a request body, or more commonly, for a response body. HTTP compression can only be done if the client sends a request header called Accept-Encoding with an encoding supported by the server:

```
GET / HTTP/1.1
```
When a response is compressed, the response will have an HTTP header saying how the body is compressed:

```
HTTP/1.1 200 OK
Content-Encoding: gzip ...
```

WAS does not natively support Content-Encoding such as gzip compression for HTTP responses (except in the proxy server or ODR).

It is recommended to do compression at the web server level (e.g. for IHS, mod_deflate or mod_gzip); however, it may be done by the application within WAS by setting the proper response header and compressing the response content using a custom servlet filter.

**Java Server Faces (JSF)**

The default setting of `org.apache.myfaces.SERIALIZE_STATE_IN_SESSION=true` in the version of MyFaces 2.0 that WAS <= 8.5.5 uses may have a significant performance overhead. The default in MyFaces 2.2 has been changed to false. However, note setting this to false causes the state to be stored in browser cookies. If the amount of state is very large, this can cause performance problems for the client-to-server interaction.

The `com.sun.faces.util.LRUMap` object can hold on to a lot of memory as this is used to hold the various JSF Views in the session. There are two types of JSF Views stored in the session. Logical Views in session and Number of views in session: A logical view is a top level view that may have one or more actual views inside of it. This will be the case when you have a frameset, or an application that has multiple windows operating at the same time. The LOGICAL_VIEW_MAP map is an LRU Map which contains an entry for each logical view, up to the limit specified by the `com.sun.faces.numberOfViewsInSession` parameter. Each entry in the LOGICAL_VIEW_MAP is an LRU Map, configured with the `com.sun.faces.numberOfLogicalViews` parameter.

By default the number of views stored for each of these maps is 15. Therefore you can see how it could end up using a lot of memory. The value of `com.sun.faces.numberOfViewsInSession` and `com.sun.faces.numberOfLogicalViews` does not have to be "4", it can whatever you feel is adequate for your application.

If either of these parameters are not in the application then it will store up to 15 views in the LRU Maps. Setting these values to something lower will result in lower memory usage by JSF.

The actual number depends on your application. Basically, if we can't find a JSF View in the session to restore we will create a new one. In general, a complex application is one that would allow a user to move back and forth to pages (think something like a wizard), or an application that contains framesets or a lot of pop up windows. For example, if a pop up window is used to fill out some information and then click submit to go back to the original page... that would require storing more views in session.

15 tends to be a high number, especially if the views are large (contains quite a lot of JSF components and their state). One thing to remember is each Logical View can contain the set number of Actual Views. That is where the idea of a frameset comes in -- one logical view for the parent page, and the actual views are the different frames.
More information and how to set the parameters:


In particular, com.sun.faces.numberOfViewsInSession and com.sun.faces.numberOfLogicalViews, potentially as low as 4 (the default for both is 15), and com.sun.face.serializeServerState=true

```xml
<context-param>
  <param-name>com.sun.faces.numberOfViewsInSession</param-name>
  <param-value>4</param-value>
</context-param>
<context-param>
  <param-name>com.sun.faces.numberOfLogicalViews</param-name>
  <param-value>4</param-value>
</context-param>
```

For general MyFaces JSF tuning guidance, see [https://wiki.apache.org/myfaces/Performance](https://wiki.apache.org/myfaces/Performance)

### MyFaces JSF Embedded JAR Search for META-INF/*.faces-config.xml

By default, the IBM Apache MyFaces JSF implementation searches JSF-enabled applications for META-INF/*.faces-config.xml files in all JARs on the application classpath. A CPU profiler might highlight such tops of stacks of this form:

```
java.util.jar.JarFile$1.nextElement
java.util.jar.JarFile$1.nextElement
org.apache.myfaces.view.facelets.util.Classpath._searchJar
org.apache.myfaces.view.facelets.util.Classpath._searchResource
org.apache.myfaces.view.facelets.util.Classpath.search
com.ibm.ws.jsf.config.resource.WASFacesConfigResourceProvider.getMetaInfConfigurationResources [...]
```

When an embedded faces-config.xml file is found, a message is written to SystemOut.log with a wsjar: prefix, so this would be a simple way to check if such embedded resource searches are needed. For example:

```
```

If your applications only use a faces-config.xml within the application itself and do not depend on embedded faces-config.xml files within JARs on the application classpath, then you can just disable these searches:

Servers > Server Types > WebSphere application servers > ${SERVER} > Web Container Settings > Web container > Custom Properties > New

Name = com.ibm.ws.jsf.disablealternatefacesconfigsearch
Value = true

If some applications do require embedded faces-config.xml files, then you can disable the search globally, but then enable the search on a per-application basis:

HTTP Sessions

The HTTP session timeout is an important factor for how much heap pressure the JVM will face. Work with the business to find the lowest reasonable value (default 30 minutes).

If a customer requires session fail over, use session persistence (database) over memory-to-memory replication. Also, with Liberty Profile v8.5 Extreme Scale is included as a component which can also be used for HTTP session replication see:
http://www.ibm.com/developerworks/websphere/techjournal/1301_ying/1301_ying.html; however, it also means managing another set of JVMs and there is risk if those JVMs should be shut down inadvertently. Consider if session failover is required as it increases complexity and decreases performance. The alternative is to affinitize requests and surgically store any critical state into a database:

If using session persistence and a customer can handle timed update semantics, use timed updates:
https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cprs_best_practice.html This is typical for very high volume websites or those with very large HTTP session sizes or both. Again, there is risk even with 10 second intervals of some data loss should a negative event occur. Therefore ensure that the business owners for the application are aware of the risk and their acknowledgment of the risk before switching to timed updates. There is also the option of manual synchronization of sessions but this does involve adding and testing additional code.

The WebSphere Contrarian: Back to basics: Session failover

"My preferred alternative is to rely not on session distribution, but instead to rely simply on HTTP server plug-in affinity to “pin” a user to an application server, although this does mean that stopping an application server JVM will result in the loss of the HttpSession object. The benefit of doing so is that there is no need to distribute the session objects to provide for HttpSession object failover when an application server fails or is stopped. The obvious downside is that a user will lose any application state and will need to log back in and recreate it, and this may or may not be acceptable for your application or business requirements. I'll mention that I’ve worked with a number of customers that in fact agree with this view and make this their standard practice."

Do not enable growable. This will allow WebSphere to ignore the max session pool size and can lead to memory exhaustion (OutOfMemoryExceptions) if capacity is not closely monitored.

Try to keep per-user session data small, ideally less than 4KB (http://www.ibm.com/developerworks/websphere/techjournal/0809_col_burckart/0809_col_burckart.html).

Session overflow of non-distributed/non-persisted sessions is generally a dangerous practice. This creates an unbounded queue for sessions, and it's rarely good to ever have unbounded queues, especially with objects that are often times quite big and long-lived. This can easily cause out of
memory errors with sudden spikes of load, and allows for simple Denial of Service (DoS) attacks, whether they be malicious or an errant script. Consider disabling session overflow for non-distributed/non-persistent sessions (by default it is disabled), and adding logic to the application to check for overflow and handle that. Then, sufficient queue tuning, session timeout tuning, and horizontal scaling should be done to support the required number of sessions. When overflow occurs for non-distributed sessions, an instance of a non-null session is returned and it is set to invalid. This can be checked by the application developer.

**Database Session Persistence**

The write frequency has a large impact on performance:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/uprs_rtuining_custom.html and


**Session Data Disappears on Fail Over**

In order for HTTP Session fail over to work properly an application has to code their Java objects properly by implementing either Serializable or Externalizable. If the developers fail to do this then when some negative event causes users to fail over to another JVM session data will simply disappear.

**Annotation Scanning**

Enterprise applications that contain many classes and are enabled for annotations processing (are not marked as "metadata-complete") take extra time to deploy. Extra time is necessary to scan application binaries for annotations that were introduced by Java EE 5. If there are no additional options to limit which classes are scanned, when scanning is enabled for a module all classes in the module must be scanned. A scan of all classes is necessary even when only a small subset of classes within a given module has annotations.


**ServletContext.getResource performance**

The Java Enterprise Edition 6 (JEE6) specification changed the behavior of ServletContext.getResource to also search for resources in META-INF/resources directories of any JAR files in /WEB-INF/lib:

"[javax/servlet/ServletContext.getResource] will first search the document root of the web application for the requested resource, before searching any of the JAR files inside /WEB-INF/lib."

(http://docs.oracle.com/javaee/6/api/javax/servlet/ServletContext.html#getResource%28java.lang.String%29)

WAS starts to implement JEE6 in version 8:
If you notice a lot of time spent in ServletContext.getResource (more specifically, com/ibm/ws/webcontainer/util/MetaInfResourcesFileUtils), or significant processing unzipping JARs with that method in the stack, and if you can confirm with your application developers that there are no resources in the JAR files in the WARs, then you can set com.ibm.ws.webcontainer.SkipMetaInfResourcesProcessing = true to revert to JEE5 behavior (https://www.ibm.com/support/knowledgectcenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rweb.jsp_staticfile.html). Related: http://www-01.ibm.com/support/docview.wss?uid=swg21671090

The custom property com.ibm.ws.webcontainer.metainfresourcescachesize, which defaults to 20, may be used to reduce META-INF/lib searching and JAR processing. If tracing is enabled with com.ibm.ws.webcontainer.util=all., a cache hit will produce the trace entry starting with "got cached META-INF name."

Starting with WAS 8.0.0.10 and 8.5.5.5, additional performance enhancements have been added: http://www-01.ibm.com/support/docview.wss?uid=swg1PI28751

**Timeouts**

In general, increasing values for timeouts or pool sizes will delay recognition of a downstream component failure, but in the case of pool sizes a larger value also provides some buffering in the event of a failure. As you can see, tuning to prevent your website from stalling in the event of a failure will require a tradeoff between increasing and decreasing various parameters. Arriving at the optimal values for your environment will require iterative testing with various settings and failure scenarios so that you (or at least your computer systems) will be prepared to fail, which in turn should help insure your success (and continued employment). (http://www.ibm.com/developerworks/websphere/techjournal/1111_webcon/1111_webcon.html#sec2)

**WebContainer Diagnostic Trace**

The following diagnostic trace can be used:

For each request, the following entries will appear in trace.log for a new connection

```
[9/26/11 16:07:30:143 PDT] 00000029 HttpInboundLi 3 Init on link: com.ibm.ws.http.channel.inbound.impl.HttpInboundLink@83d083d
  com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7
[9/26/11 16:07:30:144 PDT] 00000029 HttpInboundLi > ready:
  com.ibm.ws.http.channel.inbound.impl.HttpInboundLink@83d083d
  com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7
Entry
[9/26/11 16:07:30:144 PDT] 00000029 HttpInboundLi 3 Parsing new information:
  com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7
[9/26/11 16:07:30:146 PDT] 00000029 HttpInboundLi 3 Received request number 1
  on link com.ibm.ws.http.channel.inbound.impl.HttpInboundLink@83d083d
```
Discrimination will be called

SystemOut 0 SWAT EAR: Invoking com.ibm.Sleep by anonymous (127.0.0.1)... []

SystemOut 0 SWAT EAR: Done com.ibm.Sleep

HttpInboundLi 3 close() called:
com.ibm.ws.http.channel.inbound.impl.HttpInboundLink@83d083d
com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7

HttpInboundLi 3 Reading for another request...

For an existing connection, it will be slightly different:

HttpICLReadCa 3 complete() called:
com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7

HttpInboundLi 3 Parsing new information:
com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7

HttpInboundLi 3 Received request number 2 on link com.ibm.ws.http.channel.inbound.impl.HttpInboundLink@83d083d

HttpInboundLi 3 Discrimination will be called

HttpInboundLi 3 close() called:
com.ibm.ws.http.channel.inbound.impl.HttpInboundLink@83d083d
com.ibm.ws.channel.framework.impl.InboundVirtualConnectionImpl@6c706c7

HttpInboundLi 3 Reading for another request...

The time between the "Discrimination will be called" and "close()" lines is when the request/response is executed.

**IBM Java -Xtrace**

If you want to look at the response times of a particular Java method, and you're using the IBM JVM, then you could use -Xtrace method trace. For example, we know that all HTTP(s) requests for servlets go through javax/servlet/http/HttpServlet.service, so we could use the generic JVM argument:

```
-Xtrace:methods={javax/servlet/http/HttpServlet.service},print=mt
```

Every time this method is executed, the following entries will be written to native_stderr.log:

```
23:21:46.020*0x2b28d0018700 mt.0 >
javax/servlet/http/HttpServlet.service(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V Bytecode method, This = 2b292400fcf8
23:21:47.071 0x2b28d0018700 mt.6 <
javax/servlet/http/HttpServlet.service(Ljavax/servlet/ServletRequest;Ljavax/servlet/ServletResponse;)V Bytecode method
```

Remember that servlets can include other servlets (usually through JSPs), and the method trace entries will be properly indented, but just make sure you match the right entry and exit to get the correct elapsed time.
Method trace is more useful when you already have some idea of where the slowdown may be. For example, you can specify a list of particular business methods, and then iteratively drill down into those that are slow until you reach the slow method. This of course won't help if the problem is systemic, such as garbage collection, operating system paging, etc., since that will arbitrarily affect any methods. However, it is good at pinpointing backend slowdowns (e.g. put a method trace around database calls).

Transport Channels

Assuming IHS or ODR is proxying to WAS, change WAS to unlimited persistent incoming connections (second bullet):

The default write buffer size for HTTP requests is 32768 bytes. Responses greater than this value trigger an implicit flush, and if no content length was specified, result in the response being sent with chunked Transfer-Encoding. Setting this value much higher probably does not result in significantly fewer actual write() system calls, as the underlying OS buffers are unlikely to accept such large writes. The most interest in this property is not for performance, but as a safety net for response data being written prior to the headers being complete. Or to avoid chunked responses (one-off clients may be confused by some unexpected chunked responses, download progress cannot be estimated, etc). The equivalent buffering in Liberty (there is no Web Container channel) cannot currently be tuned.

The product web container manages all HTTP requests to servlets, JavaServer Pages and web services. Requests flow through a transport chain to the web container. The transport chain defines the important tuning parameters for performance for the web container. There is a transport chain for each TCP port that the product is listening on for HTTP requests. For example, the default HTTP port 9080 is defined in web container inbound channel.

The HTTP 1.1 protocol provides a keep-alive feature to enable the TCP connection between HTTP clients and the server to remain open between requests. By default the product closes a given client connection after a number of requests or a timeout period. After a connection is closed, it is recreated if the client issues another request. Early closure of connections can reduce performance. Enter a value for the maximum number of persistent requests to (keep-alive) to specify the number of requests that are allowed on a single HTTP connection. Enter a value for persistent timeouts to specify the amount of time, in seconds, that the HTTP transport channel allows a socket to remain idle between requests. To specify values for Maximum persistent requests and Persistent timeout (Container Settings > Web container > Web container transport chains > Select the normal inbound chain for serving requests. This chain is typically called WCInboundDefault > Click HTTP Inbound Channel (HTTP_2))

Asynchronous I/O (AIO) versus New I/O (NIO)

AIO is the default TCP transport mechanism which is a WAS feature that uses a native library on each operating system to utilize operating system features for asynchronous I/O. An alternative is NIO which is Java's built in asynchronous I/O (also uses native functions in the JVM). Historically, AIO has been disabled primarily to decrease native memory pressures on 32-bit processes running near the edge. There are no clear performance numbers comparing AIO versus NIO. Therefore, this is one of those options that you should test to see what performs better in your case. To switch to NIO: http://www-01.ibm.com/support/docview.wss?uid=swg21366862

In general, AIO should show a marginal performance improvement over NIO because it simplifies some of the selector logic and reduces thread context switching:

"Prior to WebSphere Application Server V6.x, a one-to-one mapping existed between the number of concurrent client connections and the threads in the Web container thread pool. In other words, if 40 clients were accessing an application, 40 threads were needed to service the requests. In WebSphere Application Server V6.0 and 6.1, Native IO (NIO) and Asynchronous IO (AIO) were introduced, providing the ability to scale to thousands of client connections using a relatively small number of threads. This explains why ... [you may observe] an average [number of] threads [less than the] concurrent client connection [count]."

On newer versions of Windows, AIO may have poorer performance: http://www-01.ibm.com/support/docview.wss?uid=swg21681827

AIO may report more concurrently active threads than NIO in the WebContainer thread pool because of a design difference in the way the WebContainer thread pool is used to handle network input/output. In particular, AIO runs ResultHandler Runnables in the WebContainer thread pool which may be idle in the sense that they are waiting for I/O, but are considered active by the WebContainer thread pool because they are actively waiting for AIO results. This behavior is by design and it may only be a concern if the concurrently active thread count is 90% or more of the maximum size of the thread pool. Application performance should primarily be judged by response times and throughput, not by thread pool utilization.

There are two AIO native libraries shipped with WAS: ibmaio and ibmaiodbg (e.g. .so or .dll). If the JVM is started with -DAIODebugNative=true then ibmaiodbg is loaded instead which writes additional debug tracing to traceaio.txt in the JVM's working directory (e.g. profiles/${PROFILE}). This traceaio.txt file does not wrap and cannot be enabled or disabled dynamically. In general, this should be paired with the WAS diagnostic trace

With NIO, a dedicated thread does the scheduling for the other WC threads rather than how AIO has each WC thread do scheduling as needed. This may avoid certain AIO deadlock scenarios with persistent connections where all threads are in com/ibm/ws/util/BoundedBuffer.waitForAfter
http://com/ibm/ws/http/channel/inbound/impl/HttpInboundLink.close

TCP Transport Channel

By default, the TCP transport channel allows up to 20,000 concurrently open incoming connections
Benefits of a large value are:

1. AIO/NIO intensive work (e.g. most of the time spent reading or writing HTTP responses) can process more concurrent requests.
2. There can be more keepalive connections.
3. Certain applications have many connections with little activity on each connection.
4. Other functions such as asynchronous servlets and WebSockets may require a large number of connections.

Disadvantages of a large value are:

1. If there is a backup in the application, host, or external services, too many requests can queue and increase response times without any timeout notification to end-users, unless there are timeouts in upstream proxies (for example, ServerIOTimeout in IHS).
2. The number of connections must be supported by operating system and process resource limits such (for example, on a POSIX system, every socket requires a file descriptor and thus the open file ulimit must be large enough).

503 Service Unavailable

WAS will send back a 503 in at least these situations:

- If the WAS HTTP transport channel is stopping or stopped.
- If there is an internal failure when setting up a new connection.
- If the web application containing the target servlet is stopping, stopped, restarting, uninstalled, etc.

An application may send back a 503 response itself, as can other products such as the SIP proxy, Java Proxy Server, On Demand Router, etc.

Apache HttpClient

To isolate your deployment from the OSS framework "Apache HTTP Components" provided by WAS, you would define one or more of the system properties: [link](http://www-01.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/welc6tech_opensource_isolate.html)

For example:


The input will cause the server to block all loadClass() operations on class names containing the package prefix "org.apache.http.". If you need to block getResource() operations on org/apache/http/, then you would also define property:

-Dcom.ibm.ws.classloader.server.alwaysProtectedResources=org/apache/http/

And if you need access to a subpackage of org.apache.http., or a class in org.apache.http., you could define property:
Startup

Application Startup

There is a "server.startup" thread pool (default maximum size of 3) in which applications start. If this thread pool (actually, its queue of work) is full, the following message will be printed in SystemOut.log:


However, simply increasing the server.startup thread pool may not help because large parts of application startup within the WebContainer are single threaded and you may see contention in com.ibm.ws.webcontainer.component.WebContainerImpl. A feature request to investigate this was opened: https://www.ibm.com/developerworks/rfe/execute?use_case=viewRfe&CR_ID=21322

Enabling Diagnostic Trace during Startup

The normal mechanism to enable diagnostic trace at runtime is through the Runtime tab or an MBean call; however, both are unavailable during startup. Instead, Java Surgery (https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=7d3dc078-131f-404c-8b4d-68b3b9ddd07a) may be used to call the static method to set trace dynamically and this works even during startup. Example:

```
java -jar surgery.jar -pid ${PID} -command ExecuteMethod -class com.ibm.ejs.ras.ManagerAdmin -method setTraceState -arg "+*=info:WAS.j2c=all:RRA=all:Transaction=all"
```

Java Persistence API (JPA)

OpenJPA is the default persistence provider in WAS.

Increasing the integer value of [com.ibm.websphere.jpa.entitymanager.poolcapacity] might improve performance by reducing the number of EntityManager instances that must be created. However, increasing the value affects the amount of consumed memory (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rejb_jpa_system_prop.html).

OpenJPA

If an OpenJPA application is running in a single JVM, then you may use the OpenJPA data cache: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tejb_datcacheconfig.html
Otherwise, you may use the OpenJPA second level (L2) cache provider plug-in over Dynacache:

L2 caching increases the memory consumption of the application, therefore, it is important to limit the size of the L2 cache. There is also a possibility of stale data for updated objects in a clustered environment. Configure L2 caching for read-mostly, infrequently modified entities. L2 caches are not recommended for frequently and concurrently updated entities.

If the application has a set of data that is used in a static, read-only method, like accessing basic persistent fields and persisting unidirectional relationships to a read-only type, then the WSJPA ObjectCache is a non-distributed cache of read-only entities that operates at the EntityManagerFactory object level:

To trace all OpenJPA initiated SQL statements, edit persistence.xml and add a property with name "openjpa.Log" and value "SQL=TRACE". This will go to SystemErr.log:

```
TRACE  [WebContainer : 5] openjpa.jdbc.SQL - <t 292426094, conn 2131263240>
[1730 ms] spent
```

Now look for the corresponding query, i.e. the statement corresponding to connection "conn 2131263240". The duration of the query in this case was 1730ms above.

```
TRACE  [WebContainer : 5] openjpa.jdbc.SQL - <t 292426094, conn 2131263240>
executing prestmt 393222 select doc_Id from (SELECT d.doc_Id FROM GTIMS.Doc_Component_Instance d where d.doc_Component_Id = ? and d.document_Component_Inst_Data=?
intersect SELECT d.doc_Id FROM GTIMS.Doc_Component_Instance d where d.doc_Component_Id = ? and d.document_Component_Inst_Data=?) where doc_Id!=?
[params=(long) 2, (String) -1761467286, (long) 1, (String) CORPORATION, (long) 82305]
```

Latest JPA performance options available in WAS 8.5

```
<property name="openjpa.ConnectionRetainMode" value="always"/>
<property name="wsjpa.FastPath" value="true"/>
<property name="openjpa.RestoreState" value="false"/>
<property name="openjpa.OptimizeIdCopy" value="true"/>
<property name="openjpa.ProxyManager" value="delayCollectionLoading=true"/>
```

**Dynamic Cache (Dynacache)**

"WebSphere Application Server’s Dynacache provides a general in-memory caching service for objects and page fragments generated by the server. The DistributedMap and DistributedObjectCache interfaces can be used within an application to cache and share Java objects by storing references to
these objects in the cache for later use. Servlet caching, on the other hand, enables servlet and JSP response fragments to be stored and managed by a customizable set of caching rules."


Caching the output of servlets, commands, and JavaServer Pages (JSP) improves application performance... [Dynacache] intercepts calls through a servlet service method or a command execute method, and either stores the output of the object to the cache or serves the content of the object from the dynamic cache... The dynamic cache service is enabled by default.


For command caching to operate properly, you must enable servlet caching.


There is an option called "limit memory cache size" to constrain how much memory Dynacache will use:

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/udyn_r cachescsettings.html

**Dynamic Cache Replication - Data Replication Service (DRS)**

[DRS] replicates data from the dynamic cache service across the consumers in a replication domain.

To create replication domains manually, click Environment > Replication domains in the administrative console.

To create a new replication domain automatically when you create a cluster, click Servers > Clusters > New in the administrative console.

Do not use the default value of a single replica for the Number of replicas for dynamic cache replication domains. Instead, use a full group replica for any replication domains that you configure for dynamic cache.

In the administrative console, click Servers > Server Types > WebSphere application servers > server_name > Container services > Dynamic cache service. To enable replication, select Enable cache replication. Choose a replication domain.


With replication, data is generated one time and copied or replicated to other servers in the cluster, saving time and resources. Cache replication can take on three forms:

PUSH - Send out new entries, both ID and data, and updates to those entries.

PULL - Requests data from other servers in the cluster when that data is not locally present. This mode of replication is not recommended.
PUSH/PULL - Sends out IDs for new entries, then, only requests from other servers in the cluster entries for IDs previously broadcast. The dynamic cache always sends out cache entry invalidations.

Specifically, for PUSH or PUSH/PULL, the dynamic cache broadcasts the update asynchronously, based on a timed interval rather than sending them immediately when they are created. Invalidations are sent immediately.


SHARED_PUSH policy means as an object is added to the cache, it is immediately replicated to other nodes which is expensive in terms of JVM memory usage. Instead, the SHARED_PUSH_PULL policy should be used. This means only the cache key is replicated to the other nodes, and if the object is required it is replicated on the first 'cache miss'. This is much more memory efficient at the expense of a longer response time on the first access to the cached object. As the object would only be required on failover, this would be a rare occurrence anyway. This change in caching policy should be reviewed by the application development team, and tested in a failover scenario.

The other replication mode is NOT_SHARED: "When you use the Not Shared setting, as cache entries are created, neither the cache content nor the cache IDs are propagated to other servants or servers in the replication domain. However, invalidations are propagated to other servants or servers." (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/udyn_r cachesettings.html)

There are two major types of invalidations: implicit and explicit. Implicit invalidations occur when a cache entry times out (if it has a time out) or it gets pushed out of the cache by the Least Recently Used (LRU) algorithm if the cache is full (based on the maximum cache size). Explicit invalidations occur when someone calls the DistributedMap invalidate* methods (for example, on a user logout) or through the same thing on a dependency. In some cases, implicit invalidations are not necessary to propagate, such as in large WebSphere Portal clusters: http://www-10.lotus.com/ldd/portalwiki.nsf/dx/Tuning_a_cluster_environment_%28Tuning_Guide_6.1.x%29. There are two JVM custom properties that avoid these implicit invalidations: com.ibm.ws.cache.CacheConfig.filterTimeOutInvalidation=true for the timeout case and com.ibm.ws.cache.CacheConfig.filterLRUInvalidation=true for the case when a cache is full and an entry is pushed out.

Replication type: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/udyn_s cacheinstancesettings.html

The "Both push and pull" (SHARED_PUSH_PULL) policy should be used. This means only the cache key is replicated to the other nodes, and if the object is required it is replicated on the first 'cache miss'. This is much more memory efficient at the expense of a longer response time on the first access to the cached object. If the object would only be required on failover, this would be a rare occurrence anyway. This change in caching policy should be reviewed by the application development team, and tested in a failover scenario.
**Recommending Tuning**

The following tuning is recommended as a starting point for all Dynacaches:

- `com.ibm.ws.cache.CacheConfig.useServerClassLoader=true`
- `com.ibm.ws.cache.CacheConfig.filterLRUInvalidation=true`
- `com.ibm.ws.cache.CacheConfig.filterTimeOutInvalidation=true`
- `com.ibm.ws.cache.CacheConfig.filterInactivityInvalidation=true`
- `com.ibm.ws.cache.CacheConfig.cacheEntryWindow=10`
- `com.ibm.ws.cache.CacheConfig.cacheInvalidateEntryWindow=10`
- `com.ibm.ws.cache.CacheConfig.ignoreValueInInvalidationEvent=true`

This tuning may be applied globally using the instructions under "configure the custom property globally across all configured cache instances" at [http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rdyn_tunediskcache.html](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rdyn_tunediskcache.html)

**ignoreValueInInvalidationEvent**

Specifies whether the cache value of Invalidation event is ignored. If it is true, the cache value of Invalidation event is set to NULL when the code is returned to the caller.

**propogateInvalidationsNotSharedValue**

Default set to false, which provides the best performance. If it is set to true, Dynacache will send invalidations to peer members in the cluster on cache entry insertions and updates for a NOT_SHARED cache instance. This can cause a significant performance impact.

**Disk Offload**

You might want to enable dynamic cache disk offload. This option moves cache entries that are expired from memory to disk for potential future access.


Disk cache size considerations:


Disk cache garbage collector:


**ws/WSSecureMap**

The ws/WSSecureMap Dynacache is used for horizontal security attribute propagation (web inbound security attribute propagation).

**System Dump or HPROF Heapdump Analysis**

With the [IBM Memory Analyzer Tool and the IBM Extensions for Memory Analyzer](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.memory_analyzer.doc/ae/ibmmemtshpltr.html), use the
Dynacache queries to get details of Dynacache in a **system dump**. The list of queries are:

- Review the number of entries, cache size, and hit ratio:

  ```
  * dynacache_instances -instancename default
  
<table>
<thead>
<tr>
<th>Class Name</th>
<th>Shallow Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Regex&gt; com.ibm.ws.cache.Cache @ 0x3ed3390</td>
<td>328</td>
</tr>
</tbody>
</table>
  
<table>
<thead>
<tr>
<th>Retained Heap</th>
<th>Instance Name</th>
<th>Entries</th>
<th>Size Limit</th>
<th>Percent Full</th>
</tr>
</thead>
<tbody>
<tr>
<td>21,232</td>
<td>default</td>
<td>12</td>
<td>2,000</td>
<td>1%</td>
</tr>
</tbody>
</table>

- A high number of misses could mean that the cache size is too small, there are many invalidations, there is a get-check-update pattern without warmup/pre-loading, etc.

**Servlet Caching**

Servlet caching may cause a significant throughput improvement. In one benchmark, 30-60% ([http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html](http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html)).

Use this task to define cacheable objects inside the cachespec.xml, found inside the web module WEB-INF or enterprise bean META-INF directory... [or] you can save a global cachespec.xml in the application server properties directory, but the recommended method is to place the cache configuration file with the deployment module.
In situations where there is a global cachespec.xml file in the application server properties directory, and a cachespec.xml file in an application, the entries in the two cachespec.xml files are merged. If there are conflicting entries in the two files, the entries in the cachespec.xml file that is in the application override the entries in the global cachespec.xml file for that application.

To cache an object, WebSphere® Application Server must know how to generate unique IDs for different invocations of that object. The <cache-id> element performs that task. Each cache entry can have multiple cache-ID rules that run in order until either a rule returns a cache-ID that is not empty or no more rules remain to run. If no cache-ID generation rules produce a valid cache ID, then the object is not cached.

Use dependency ID elements to specify additional cache group identifiers that associate multiple cache entries to the same group identifier. The dependency ID is generated by concatenating the dependency ID base string with the values returned by its component elements. If a required component returns a null value, then the entire dependency ID is neither generated nor.

Invalidate other cache entries as a side effect of this object start, if relevant. You can define invalidation rules in exactly the same manner as dependency IDs... The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules run separately.

The dynamic cache reloads the updated file automatically. If you are caching static content and you are adding the cache policy to an application for the first time, you must restart the application. You do not need to restart the application server to activate the new cache policy.

When new versions of the cachespec.xml are detected, the old policies are replaced. Objects that cached through the old policy file are not automatically invalidated from the cache; they are either reused with the new policy or eliminated from the cache through its replacement algorithm.


Full cachespec.xml schema:

The <timeout> is specified in seconds. If a timeout is not specified, then the cache entry does not expire. However, in both cases, a cache entry may be evicted or invalidated either explicitly, by invalidation rules, or by the Least Recently Used (LRU) algorithm when the cache is full.

Before WAS 9, if servlet caching is enabled in an application server, all requests pass through Dynacache, even if there is no chance they will be cached. Starting in WAS 9, only requests that have a
context root associated with a cachespec.xml flow through Dynacache. This means that using a global cachespec.xml in \${WAS}/profiles/${PROFILE}/properties only works for applications that have a cachespec.xml in the WAR's WEB-INF. To "defer" to a global cachespec.xml, a "dummy" cachespec.xml may be placed in the application such as:

```xml
<?xml version="1.0" ?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
  <cache-entry>
    <class>servlet</class>
    <name>/dummy</name>
  </cache-entry>
</cache>
```

Example custom ID generators: [https://github.com/kgibm/WASServletCachingIDGenerators](https://github.com/kgibm/WASServletCachingIDGenerators)

**Servlet Caching Example**

Suppose that a servlet manages a simple news site. This servlet uses the query parameter "action" to determine if the request views (query parameter "view") news or updates (query parameter "update") news (used by the administrator). Another query parameter "category" selects the news category. Suppose that this site supports an optional customized layout that is stored in the user's session using the attribute name "layout". Here are example URL requests to this servlet:

- [http://yourhost/yourwebapp/newscontroller?action=view&category=sports](http://yourhost/yourwebapp/newscontroller?action=view&category=sports) (Returns a news page for the sports category)
- [http://yourhost/yourwebapp/newscontroller?action=view&category=money](http://yourhost/yourwebapp/newscontroller?action=view&category=money) (Returns a news page for the money category)

Define the <cache-entry> elements that are necessary to identify the servlet. In this case, the URI for the servlet is "newscontroller", so this is the cache-entry <name> element. Because this example caches a servlet or JavaServer Pages (JSP) file, the cache entry class is "servlet".

Define cache ID generation rules. This servlet caches only when action=view, so one component of the cache ID is the parameter "action" when the value equals "view". The news category is also an essential part of the cache ID. The optional session attribute for the user's layout is included in the cache ID.

Define dependency ID rules. For this servlet, a dependency ID is added for the category. Later, when the category is invalidated due to an update event, all views of that news category are invalidated.

Define invalidation rules. Because a category dependency ID is already defined, define an invalidation rule to invalidate the category when action=update. To incorporate the conditional logic, add "ignore-value" components into the invalidation rule. These
components do not add to the output of the invalidation ID, but only determine whether or not the invalidation ID creates and runs.

<cache-entry>
  <name>newscontroller</name>
  <class>servlet</class>
  <cache-id>
    <component id="action" type="parameter">
      <value>view</value>
      <required>true</required>
    </component>
    <component id="category" type="parameter">
      <required>true</required>
    </component>
    <component id="layout" type="session">
      <required>false</required>
    </component>
  </cache-id>
  <dependency-id>category</dependency-id>
  <invalidation>category</invalidation>
</cache-entry>


**Servlet Caching by User**

Be careful in building your cache ID if cached objects may be user-specific. In such a case, you can use some user-identifiable component for the cache ID such as the JSESSIONID:

<cache-entry>
  <class>servlet</class>
  <name>/forward.do</name>
  <cache-id>
    <property name="EdgeCacheable">true</property>
  </cache-id>
</cache-entry>
Monitoring

"Use the administrative console to install the cache monitor application from the app_server_root/installableApps directory. The name of the application is CacheMonitor.ear... you can access the cache monitor using http://your_host_name:your_port_number/cachemonitor" (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tdyn_servletmonitor.html).

Monitor the cache hit ratio:

Use JMX:

If javacores or other indicators show a lot of JMX activity causing performance issues, use com.ibm.ws.management.connector.soap.logClientInfo=true

Example in dmgr SystemOut.log:


Object Request Broker (ORB) and Remote Method Invocation (RMI)

For IBM JVMs, additionally see the ORB section in the IBM Java chapter.

ORB pass by reference (com.ibm.CORBA.iop.noLocalCopies) may cause a significant throughput improvement. In one benchmark, 50-60%
The Object Request Broker (ORB) pass by reference option determines if pass by reference or pass by value semantics should be used when handling parameter objects involved in an EJB request. This option can be found in the administrative console by navigating to Servers => Application Servers => server_name => Object Request Broker (ORB). By default, this option is disabled and a copy of each parameter object is made and passed to the invoked EJB method. This is considerably more expensive than passing a simple reference to the existing parameter object.

To summarize, the ORB pass by reference option basically treats the invoked EJB method as a local call (even for EJBs with remote interfaces) and avoids the requisite object copy. If remote interfaces are not absolutely necessary, a slightly simpler alternative which does not require tuning is to use EJBs with local interfaces. However, by using local instead of remote interfaces, you lose the benefits commonly associated with remote interfaces, location transparency in distributed environments, and workload management capabilities.

The ORB pass by reference option will only provide a benefit when the EJB client (that is, servlet) and invoked EJB module are located within the same classloader. This requirement means that both the EJB client and EJB module must be deployed in the same EAR file and running on the same application server instance. If the EJB client and EJB modules are mapped to different application server instances (often referred to as split-tier), then the EJB modules must be invoked remotely using pass by value semantics. (http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html)

Set com.ibm.CORBA.ServerSocketQueueDepth to 511 (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tprf_tuneappserv.html). If this value is reached, subsequent connection attempts will receive connection refused errors after a connection timeout period (and potentially implicit retries).

The thread pool size is dependent on your workload and system. In typical configurations, applications need 10 or fewer threads per processor. (Servers > Server Types > Application servers > server_name > Container services > ORB service > Thread pool)

Each inbound and outbound request through the ORB requires a thread from the ORB thread pool. In heavy load scenarios or scenarios where ORB requests nest deeply, it is possible for a Java virtual machine (JVM) to have all threads from the ORB thread pool attempting to send requests. Meanwhile, the remote JVM ORB that processes these requests has all threads from its ORB thread pool attempting to send requests. As a result, progress is never made, threads are not released back to the ORB thread pool, and the ORB is unable to process requests. As a result, there is a potential deadlock. Using the administrative console, you can adjust this behavior through the ORB com.ibm.websphere.orb.threadPoolTimeout custom property.


Monitor and tune the ORB service thread pool:
Monitor and tune the connection cache size (com.ibm.CORBA.MaxOpenConnections): http://www.ibm.com/support/knowledgecenter/SSAW57_8.0.0/com.ibm.websphere.nd.doc/info/ae/orb_tims.html. Ideally, this should be greater than or equal to the maximum number of concurrent connections, but not so large as to cause too many threads (or in such a case, JNI Reader Threads could be used instead).

By default, the option to "prefer local" (meaning to prefer sending requests to EJBs on the same node, if available) is enabled; however, the deployment manager must be running for it to function: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/urun_rwlm_cluster_create1.html?lang=en

Running with Java security enabled will reduce performance. For example: http://www-01.ibm.com/support/docview.wss?uid=swg21661691

**EJBs**

If the Performance Monitoring Infrastructure (PMI) counters show a high rate of ejbStore methods being called, then the EJB container cache size may need to be increased: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rprf_ejbcontainer.html

Run the EJB Cache trace to ensure the cache sizes are tuned optimally: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tejb_tunecash.html

If there is significant heap pressure from stateful session beans (check heapdumps), consider specifying a timeout that the application can handle using 

If PMI shows that most bean instances are being used in the pool, consider increasing the pool size for that application: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rprf_ejbcontainer.html. For example, com.ibm.websphere.ejbcontainer.poolSize="*=,3000"

**JNI Reader Threads**

In general, switching to JNI Reader Threads is only recommended when a very large number of concurrent users/connections is required. Instead of the default one thread per connection (and each client will have at least 2-3 connections: one for bootstrap, one for the listener, and potentially one for TLS), JNI reader threads only require a handful of threads (usually 4 is enough, which is the default), each one of which handles up to 1,024 connections simultaneously using asynchronous I/O.

By default, the ORB uses a Java thread for processing each inbound connection request it receives. As the number of concurrent requests increases, the storage consumed by a large number of reader threads increases and can become a bottleneck in resource-constrained
environments. Eventually, the number of Java threads created can cause out-of-memory exceptions if the number of concurrent requests exceeds the system's available resources.

To help address this potential problem, you can configure the ORB to use JNI reader threads where a finite number of reader threads, implemented using native OS threads instead of Java threads, are created during ORB initialization. JNI reader threads rely on the native OS TCP/IP asynchronous mechanism that enables a single native OS thread to handle I/O events from multiple sockets at the same time. The ORB manages the use of the JNI reader threads and assigns one of the available threads to handle the connection request, using a round robin algorithm. Ordinarily, JNI reader threads should only be configured when using Java threads is too memory-intensive for your application environment.

Each JNI thread can handle up to 1024 socket connections and interacts directly with the asynchronous I/O native OS mechanism, which might provide enhanced performance of network I/O processing.

If JNI Readers Threads are enabled, the default number (com.ibm.CORBA.numJNIReaders) is 4 which can handle up to 4,096 concurrent connections:


Workload Management (WLM)

"Multiple application servers can be clustered with the EJB containers, enabling the distribution of enterprise bean requests between EJB containers on different application servers... EJB client requests are routed to available EJB containers in a round robin fashion based on assigned server weights." (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/crun_srvgrp.html?lang=en)

WLM balances requests in the form of method calls/invocations. The "pattern problem" occurs when there is a pattern of method calls that correlates with the number of cluster members. For example, if there are two cluster members and an even number of method calls such as "create" and "invoke," it's possible that all the lightweight create requests execute on one server, and the heavyweight invoke requests execute on the other server. In that case the "load" on the servers (for example, measured in CPU utilization) is not equal among the servers. Workarounds to this problem include 1) changing the number of cluster members (for example, from even to odd), and 2) adjusting the weights of the cluster members to non-equal values (typically recommended for normalization are cluster weights of 19 and 23).

Java Naming and Directory Interface (JNDI)

By default the JNDI naming caches are unbounded and persist for the life of the JVM. There is one cache per provider URL. If applications use a large variety of names or large named objects, then the
caches may use significant amounts of memory. Each cache can be made to timeout (on next access) using the `-Dcom.ibm.websphere.naming.jndicache.maxcachelife=$minutes` property: [http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rnam_jndi_settings.html?cp=SSAW57_8.5.5&lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rnam_jndi_settings.html?cp=SSAW57_8.5.5&lang=en). The caches can be completely disabled with `-Dcom.ibm.websphere.naming.jndicache.cacheobject=none`. These properties can be placed into the properties Hashtable used in creating the InitialContext.

You can find the size of all JNDI caches by gathering a heapdump or coredump. Open the IBM Memory Analyzer Tool, and click Open Query Browser > Show Retained Set. For the class row, type `com.ibm.ws.naming.jcache.Cache` and click OK. Review the sum of shallow heaps in the bottom right.

**InitialContext**

A `javax.naming.InitialContext` is the starting point to perform naming operations. There is significant processing in creating an InitialContext, so it is recommended to cache them. However, an InitialContext is not thread safe:

An InitialContext instance is not synchronized against concurrent access by multiple threads. ([http://docs.oracle.com/javase/8/docs/api/javax/naming/InitialContext.html](http://docs.oracle.com/javase/8/docs/api/javax/naming/InitialContext.html))

It is recommended to use ThreadLocals to create InitialContexts once. For example:

```java
private final ThreadLocal<InitialContext> jndiContext = new ThreadLocal<InitialContext>() {
    protected InitialContext initialValue() {
        try {
            final InitialContext context = new InitialContext();
            return context;
        } catch (NamingException e) {
            throw new RuntimeException(e);
        }
    }
};
```

InitialContexts are often used to bind once at application startup (in which case a thread local is not needed); however, it is common practice to catch exceptions on object invocations and re-lookup a resource at runtime, in which case ThreadLocals should be used to avoid the cost of creating InitialContexts.

**Message Driven Beans (MDBs)**

You can choose Activation Specifications or Listener Ports for handling MDBs:

- Listener ports are "stabilized" and you should use Activation Specifications instead: [https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tmji_adm32.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tmji_adm32.html)
- Listener Ports follow the principle of the Application-Service-Facility (ASF) part of the JMS specification and thus they poll for messages from the destinations. Activation Specifications on the other hand work on the principle of callbacks and notifications, so there is no polling involved.
- Listener Ports are not portable, whereas Activation Specifications are portable across JEE
environments.

• Activation Specifications are better in performance due to the nature of callbacks: the moment a message is available on the destination the message provider notifies the consumers. For Listener Ports, the thread has to constantly poll for messages. Once a message is found, it has to spawn a new thread, pass the message reference to the other thread so that the other thread can actually do a get.

**Activation Specifications**

MDB concurrency is the primary tuning variable, along with the thread pool on which MDBs execute:

• SIBus MDB concurrency set with: Maximum Concurrent MDB invocations per endpoint. Updates to this value require a restart of the messaging cluster.
• WebSphere MQ Messaging Provider concurrency set with: Advanced Properties > Maximum Server Sessions

Pause an Activation Specification using `wsadmin -lang jython`:

```
AdminControl.invoke(AdminControl.queryNames("*:type=J2CMessageEndpoint,ActivationSpec=jms/testactivationspec1,*"), "pause")
J2CA0524I: The Message Endpoint ... is deactivated.
```

Resume an Activation Specification using `wsadmin -lang jython`:

```
AdminControl.invoke(AdminControl.queryNames("*:type=J2CMessageEndpoint,ActivationSpec=jms/testactivationspec1,*"), "resume")
J2CA0523I: The Message Endpoint ... is activated.
```

Get status of an Activation Specification using `wsadmin -lang jython`:

```
getStatus:
AdminControl.invoke(AdminControl.queryNames("*:type=J2CMessageEndpoint,ActivationSpec=jms/testactivationspec1,*"), "getStatus")
1 (Active), 2 (Inactive), 3 (Stopped).
```

**Listener Ports**

MDB concurrency is the primary tuning variable, along with the thread pool on which MDBs execute:

• MDB concurrency set with: Maximum Sessions

Stop a listener port using `wsadmin -lang jython`:

```
AdminControl.invoke(AdminControl.queryNames("*:type=ListenerPort,name=LPNAME,process=server1,*"), "stop")
WMSG0043I: MDB Listener... stopped...
```

Start a listener port using `wsadmin -lang jython`:

```
AdminControl.invoke(AdminControl.queryNames("*:type=ListenerPort,name=LPNAME,process=server1,*"), "start")
WMSG0042I: MDB Listener... started successfully...
```

Print if a listener port is started or not via `wsadmin -lang jython`:

```
Print if a listener port is started or not via `wsadmin -lang jython`:
```
AdminControl.getAttribute(AdminControl.queryNames("*:type=ListenerPort,name=LPNAME,process=server1,*"), "getStatus")
Returns true or false

Background on how listener ports work:

- How message-driven bean listener ports use the connection pool:
- How the maximum sessions property on the listener port affects WebSphere Application Server performance:
- Get the most out of high performance message-driven beans and WebSphere Application Server:

**Service Integration Bus (SIB)**

The Service Integration Bus is a pure Java JMS provider built into WAS:

- Bus: Group of messaging engines
- Bus Member: Hosts messaging engine
- Messaging Engine (ME): Handles destinations (queues, topics), connections, and messages
- ME Cluster Policy:
  - High availability: ME(s) will failover to other available cluster members
  - Scalability: Each cluster member runs an ME
  - Both: Each ME may failover to one other cluster member

For Activation Specifications, messages are processed on the SIBJMSRAThreadPool thread pool. Network communication is processed on the SIBFAP*ThreadPool thread pool. Therefore, the sum of the maximum concurrent invocations per endpoint for all Activation Specifications should be less than or equal to the maximum size of the SIBJMSRAThreadPool thread pool.

For listener ports, messages are processed on the MessageListenerThreadPool thread pool. Therefore, the sum of the maximum sessions for all listener ports should be less than or equal to the maximum size of the MessageListenerThreadPool thread pool.

SIB properties set in the administrative console take precedence over properties set in the sib.properties file.

Tune the maximum concurrent endpoints and maximum batch sizes:

Set IBM_CLUSTER_ENABLE_ACS_DELAY_POSTING=true

Consider increasing various data buffer sizes:
There are several factors that affect SIBus performance. The more destinations there are hosted on a messaging engine, the longer it takes for the engine to start... If the same number of destinations are apportioned over more than one bus member, the startup time improves considerably. However, the drawback is that there are more network connections between the bus members, more overall resource usage, and that the configuration becomes more complex.

If you have many disjoint destinations in your bus being used by different applications, consider creating different buses

You must tune the environment so that messages are consumed at a rate slightly higher than the rate that they are produced. If the producer produces messages at a faster rate, the messages will overload the bus.


On z/OS, the control region adjunct (CRA) address space runs SIBus messaging engines and the MDBs run in the servants (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/cprf_tunezmdb.html?cp=SSAW57_8.5.5%2F3-2-9-2-7).

If you are using durable subscriptions, explicitly set the activation specification configuration in the Administrative Console within the cluster that hosts the durable subscription home ME to Target type = Messaging engine name, Target Significance = Required, and Target = Durable subscription home messaging engine. Otherwise, remote GETs may occur in some situations (particularly failover) and they are pointless overhead (both CPU and memory) for durable subscriptions. Background: http://www.ibm.com/developerworks/websphere/library/techarticles/0811_roberts/0811_roberts.html

If SIB is using a database message store, after SIB has started, it checks the database to ensure a lock every 20 seconds. If this check fails due to a fatal database exception and jdbcFailoverOnDBCConnectionLoss = true (default), then this will lead to the JVM shutting itself down (through an HAM panic) to force a SIB failover to another JVM. If jdbcFailoverOnDBCConnectionLoss = false, SIB will continue trying to get the lock every 20 seconds (during which there is a potential for data loss). If another highly available cluster member is running, the high availability manager will automatically start the messaging engine on another running server. During SIB startup, the properties jdbcInitialDatasourceWaitTimeout (default 15 minutes) and jdbcStaleConnectionRetryDelay (default 2 seconds) are used to retry errors during startup: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tjm_tunedbconn.html.

Message Reliability
The SIBus provides five different levels of reliability.

- Best effort non-persistent
- Express non-persistent
• Reliable non-persistent
• Reliable persistent
• Assured persistent

Persistent messages are always stored to some form of persistent data store, while non-persistent messages are generally stored in volatile memory. There is a trade-off here between reliability of message delivery and the speed with which messages are delivered. The further the reliability level decreases, the faster messages can be processed (http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html)

Non-persistent message reliability may cause a significant throughput improvement. In one benchmark, 29% (http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html).

If you are using mediations and not using assured persistent messages, consider skipping the well formed check: https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tjp0032_.html

Message Store


• Local Derby database data store: With this option, a local, in-process Derby database is used to store the operational information and messages associated with the messaging engine. Although convenient for development purposes, this configuration uses valuable cycles and memory within the application server to manage the stored messages.

• File-based data store: (default) If the message engine is configured to use a file-based data store, operating information and messages are persisted to the file system instead of a database. This performs faster than the local Derby database and, when a fast disk such as a redundant array of independent disks (RAID) is used, can perform just as fast as a remote database. The test results shown below did not use a RAID device for the file-based data store and do not reflect this additional improvement.

• Remote database data store: In this configuration, a database residing on a remote system is configured to act as the message engine data store. This frees up cycles for the application server JVM process that were previously used to manage the Derby database or file-based stores, enabling a more performant, production level database server (such as IBM DB2® Enterprise Server) to be used. One technical advantage of using a database for the data store is that some J2EE™ applications can share JDBC connections to benefit from one-phase commit optimization. For more information see information on sharing connections to benefit from one-phase commit optimization. File store does not support this optimization.

Using a remote data store may cause a significant throughput improvement. In one benchmark, 55% (http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html).

IBM DB2: "To get the best performance from messages in the 3 KB to 20 KB range, you should consider putting the SIBnmm tables into a tablespace with 32 KB pages and adjusting the column width of the VARCHAR column to 32032 bytes." (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cjm0470_.html)
The File store log directory can be specified during the creation of an SIBus member using the -logDirectory option in the AdminTask addSIBusMember command or via the administration console SIBus Member creation panels. This should be moved to fast disks.

If statistics suggest a concurrency bottleneck on the SIBnnn tables for a data store, you might try to solve the problem by increasing the number of tables:


Monitoring

The CWSID0016I message indicates the state of messaging engines. For example:

```
000000fe SibMessage    I   [:] CWSID0016I: Messaging engine ${NAME} is in state Starting.
000000fe SibMessage    I   [:] CWSID0016I: Messaging engine ${NAME} is in state Joining.
000000fe SibMessage    I   [:] CWSID0016I: Messaging engine ${NAME} is in state Joined.
000000fe SibMessage    I   [:] CWSID0016I: Messaging engine ${NAME} is in state Started.
000000fe SibMessage    I   [:] CWSID0016I: Messaging engine ${NAME} is in state Stopping.
000000fe SibMessage    I   [:] CWSID0016I: Messaging engine ${NAME} is in state Stopped.
```

The printSIBusSummary.py script may be used to print the details of all messaging engines in all buses: http://www-
01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/tjo0060_.html?cp=SSAW57_8.5.5&lang=en

Important PMI metrics:

- JCA Connection Factory: PoolSize-FreePoolSize, UseTime, WaitTime
- SIB Service > SIB Messaging Engines > * > Destinations > Queues
  - AvailableMessageCount: Number of messages available for consumption from this queue
  - AggregateMessageWaitTime: Total amount of time spent in the bus by messages consumed from this queue
  - UnavailableMessageCount: Number of messages on this queue but not available for consumption
  - TotalMessagesProducedCount: Total number of messages produced to this queue
  - LocalConsumerCount: Number of local consumers currently attached to this queue
  - LocalProducerCount: Number of local producers currently attached to this queue
  - LocalMessageWaitTime: Total amount of time spent on this queue by messages consumed from this queue
  - TotalMessagesConsumedCount: Total number of messages consumed from this queue
- Enterprise Beans
  - MessageBackoutCount: The number of backed out messages that failed to be delivered to the onMessage method of the bean (applies to: message-driven beans).
  - MessageCount: The number of messages delivered to the onMessage method of the bean
(applies to: message-driven beans).

- **MethodResponseTime**: The average response time in milliseconds on the remote methods of the bean.
- **ActiveMethodCount**: Average concurrently actively called methods.

**SIB Service > SIB Messaging Engines > * > Storage Management > Data Store**

- **JDBCTransactionTime**: Total execution time of internal batches
- **PersistentDispatcherAvoidanceCount**: Measures the number of operations on reliable persistent data dispatched for writing to the data store but whose writing was subsequently unnecessary.

**SIB Service > SIB Messaging Engines > * > Storage Management > File Store**

- **FileStoreLogSpace**: Space in bytes left in the file store log
- **FileStorePermanentObjectStoreSpace**: Space in bytes left in the file store permanent store

**Message Visibility/Message Gathering**

Message visibility/message gathering may be used to consume messages from all available queue points of a destination. This may be useful when cluster members have different configurations or processing speeds; however, message visibility itself has a very high performance overhead ([http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/cjt0024_.html?lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/cjt0024_.html?lang=en)). In general, using a single, highly available, clustered destination will probably perform better in the case of differing node performance.

**Administrative Console Monitoring**

In recent versions of WAS, the administrative console provides basic monitoring on the Runtime tab of the Messaging Engine.

**Publication point queue depth:**

<table>
<thead>
<tr>
<th>Resource</th>
<th>Queue Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Topic Space</td>
<td>0</td>
</tr>
<tr>
<td>TestTopicSpace2</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>2</td>
</tr>
</tbody>
</table>

**Subscription queue depth:**
Service Integration Bus Destination Handler

The IBM Service Integration Bus Destination Handler tool is a free download that can view, move, copy, delete and restore messages:  [http://www-01.ibm.com/support/docview.wss?uid=swg24021439](http://www-01.ibm.com/support/docview.wss?uid=swg24021439)

Service Integration Bus Explorer

SIB Explorer is a free download to monitor SIBus: [https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=fe21f954-66dd-40c2-bfc4-45eb0b7d51eb](https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=fe21f954-66dd-40c2-bfc4-45eb0b7d51eb)

Service Integration Bus Performance

SIB Performance is a free download to monitor SIBus performance: [https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=13a714e5-6379-4325-aeee-c0e88ec3d15b](https://www.ibm.com/developerworks/mydeveloperworks/groups/service/html/communityview?communityUuid=13a714e5-6379-4325-aeee-c0e88ec3d15b)
WebSphere MQ Messaging Provider

Messages are processed on the WMQICAResourceAdapter thread pool. Therefore, the sum of the maximum concurrent invocations per endpoint for all Activation Specifications should be less than or equal to the maximum size of the WMQICAResourceAdapter thread pool.

MDB Response Times

Request Metrics (covered earlier) can be used to track the response times of individual MDB transactions:

1. Ensure "Prepare Servers for Request metrics collection" is checked
2. Select "Custom" for "Components to be instrumented" and select "JMS" and any other relevant components
3. Set "Trace level" to "Hops"
4. Check "Standard Logs"

Simply save and synchronize the changes and request metrics is dynamically enabled.

Here is an example where a servlet calls a stateless session bean which puts a message on a queue. Then an Activation Specification reads from this service integration bus queue to parse the message and sleep one second. Here is the output in SystemOut.log:

```
[4/17/14 8:56:49:563 PDT] 00000021 PmiRmArmWrapp I PMRM0003I:
    parent:ver=1,ip=127.0.0.1,time=1397749842692,pid=21621,reqid=8,event=1 -
    current:ver=1,ip=127.0.0.1,time=1397749842692,pid=21621,reqid=8,event=1
type=URI detail=/TestAdvancedWeb/Test elapsed=78
[4/17/14 8:56:49:563 PDT] 00000028 PmiRmArmWrapp I PMRM0003I:
    parent:ver=1,ip=127.0.0.1,time=1397749842692,pid=21621,reqid=8,event=1 -
    current:ver=1,ip=127.0.0.1,time=1397749842692,pid=21621,reqid=4097,event=1
type=JMS detail=queue1 elapsed=1034
```

The "type=JMS" line indicates that an MDB has finished processing, the detail field shows the WAS queue name, and the elapsed field shows it took 1034ms. Interestingly, I also had the Servlet component enabled in request metrics, and you can see that the "parent" of the JMS line is the "current" of the servlet line, which means the correlation crosses some boundaries and allows us to know that this message was processed from the same transaction as that particular servlet invocation. This is very useful for tracking asynchronous requests.

JMS Connections Explained

When an application obtains a J2EE Managed connection, which includes JMSConnections obtained from a (JCA managed) connection factory looked up in the J2EE app server's naming context, the J2C connection manager owns/manages the JMS Connection, what the application is given is a wrapper (an instance of the JMSCloneConnectionHandle class which implements the JMS Connection interface). It is this connection handle that is 'closed' by the container/J2C - once it is closed further attempts to use the connection handle (for example to create a session) will fail with javax.jms.IllegalStateException: Connection closed
Once closed the connection handle cannot be re-opened.

When looking in the connection pool what they see are the JMS Connections that the J2C connection manager is managing (technically it actually manages javax.resource.spi.ManagedConnection objects which themselves are wrappers to the actual JMS Connection, and the connection handles). JMS Connections are not closed when the close is called (on the connection handle) but returned to the pool (...for unshared - for shared they are available for reuse until returned to the freepool when the transaction context ends).

NOTE that the handle is closed in compliance with the J2EE connector architecture specification. The close of the handle is NOT part of the transaction context (JTA or LTC) ending but performed by the container in concert with J2C as part of application component instance lifecycle management as per the JCA spec. While it is absolutely correct that JMS Connections are required by the JMS specification to be thread safe and are non-Transactiona, they are still managed connections in a J2EE app server environment.

In other words, the issue here is NOT that managed JMS Connections cannot be reused under different LTCs, it is that the handles to the managed connections are closed, rendering them unusable, as part of the interaction between J2C and the container managing the lifecycle of the application component instance. It just looks like an LTC association is causing the issue (which is indeed non-sensical for JMS Connections and no such association exists in WAS) because LTCs tend to be associated with invocations of app component instances which themselves are associated with their lifecycle. (I note this contradicts an earlier update regarding connection handles being closed at LTC completion.

However, there is therefore scope for caching the connection handle inside a stateful session bean since the bean instance will persist over invocations of the SFSB - however passivation of the SFSB would need to be accounted for, as would connection failure handling.

The above approach is generally discouraged since J2C is managing connections and it is generally a bad idea for two entities to attempt to manage the same resource - which effectively is what the app would be attempting to do by caching+reusing the connection. It is also worth noting that JMS connections themselves may not map one-to-one with the actual TCP connections to the QMGR and a large number of them may not pose a resource issue, for example WMQ multiplexes a configurable number of multiple JMS connections and sessions down the same TCP connection though this will be JMS provider specific.

An alternative is to use J2SE JMS. Using this alternative means using a non JCA managed connection factory which will produce non-managed connections and non-managed sessions. Management (caching/reuse/threading/connection failure etc) of the connections/sessions etc is then the sole responsibility of the application. Any work performed against the sessions would not be enlisted with transactions (LTC or JTA) - they would behave just as they would in a J2SE environment.

**Web Services**


rtprop.html

Set `-Dcom.ibm.ws.websvcs.getJAXBContext.cacheClassList.persist=true`

On WAS >= 8.5.5.2, if applicable, use `com.ibm.websphere.webservices.jaxwsOptimizeLevelOne=true`

If you have web services applications that use transport level security for XML encryption or digital signatures, use the unrestricted JCE policy files:

JAX-RPC (not JAX-WS) web services support response compression using `com.ibm.websphere.webservices.http.responseContentEncoding`

If sending web services requests from an MDB, use `com.ibm.ws.websvcs.transport.jms.cacheReplyQCF=true` (http://www-01.ibm.com/support/docview.wss?uid=swg1PM93004)

If using JAX-WS on WAS >= 8.5.5.2, consider setting `-DcacheTransformerFactory=true` (http://www-01.ibm.com/support/docview.wss?uid=swg1PI06819).


WSPerf Tool
https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/Troubleshooting_Web_Services_Performance_Problems_In_IBM_WebSphere_Application_Server_With_Web_Services_Performance_WSPerf_Trace?lang=en

Inbound Web Services Processing


Preferring Local Execution

If the web services client is running in the same JVM as the web service target, consider using the optimized local communication path:

To improve performance, there is an optimized communication path between a web services client application and a web container that are located in the same application server process. Requests from the web services client that are normally sent to the web container using a network connection are delivered directly to the web container using an optimized local path. The local path is available because the web services client application
and the web container are running in the same process.

The optimized local communication path is disabled by default. You can enable the local communication path with the enableInProcessConnections custom property. Before configuring this custom property, make sure that you are not using wildcards for host names in your web container end points. Set this property to true in the web container to enable the optimized local communication path. When disabled, the web services client and the web container communicate using network transports.


Outbound Connection Cache

For performance reasons, ensure that the com.ibm.websphere.webservices.http.maxConnection custom property is at least half the size of the maximum number of threads in the web container thread pool. The default size for the web container thread pool is 50. As a result, the default size of the com.ibm.websphere.webservices.http.maxConnection property is set to 25 and 50 for JAX-RPC and JAX-WS, respectively. You can adjust the setting for com.ibm.websphere.webservice.http.maxConnection upwards from this initial value, as required, to better utilize the threads.


The outbound connection cache does not have PMI monitoring but does have a lightweight monitor trace:

- JAX-RPC
  - com.ibm.ws.webservices.engine.transport.channel.Monitor=all=enabled
- JAX-WS
  - com.ibm.ws.webservice.http.maxConnection

Details about this monitoring capability may be found here: http://www-01.ibm.com/support/docview.wss?uid=swg1PK77273

Web Services Response Caching

Enabling the web services client cache is an option to improve the performance of your system by using the dynamic cache service to save responses from remote web services for a specified amount of time... You enable the web services caching by enabling the dynamic cache service and servlet caching. After a response is returned from a remote web service, the response is saved in the client cache on the application server. Any identical requests that are made to the same remote web service are then responded to from the cache for a specified period of time. The web services client cache relies primarily on time-based invalidations because the target web service can be outside of your enterprise network and unaware of your client caching. Therefore, you can specify the amount of time in the
Using Dynacache to cache web service responses:

Asynchronous Beans

- Legacy WAS work asynchronous beans implement `com.ibm.websphere.asynchbeans.Work` and are run asynchronously by a WAS WorkManager (which manages a set of threads) through the `startWork` method call.
- CommonJ work asynchronous beans implement `commonj.work.Work` and are run asynchronously by a CommonJ WorkManager (which manages a set of threads) through the `schedule` method call.
- Timer listener asynchronous beans implement `commonj.timers.TimerListener` and are run asynchronously by a WAS timer manager that implements `commonj.timers.TimerManager`. These timers are used to schedule future work and are appropriate for managed JEE environments, unlike instances of `java.util.Timer`.
- Alarm listener asynchronous beans implement `com.ibm.websphere.asynchbeans.AlarmListener` and are run asynchronously by a WAS alarm manager (which manages a set of threads). These alarms are used to schedule future work.

Work Manager

If a non-zero "work timeout" is specified and if the time the work bean has been queued for execution plus the execution time exceeds the work timeout, then the WorkManager will call `release()` on the work bean. (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/asyncbns/tasks/tasb_workmanager.html?lang=en)

Intelligent Management

Intelligent Management (IM) was formerly a separate product called WebSphere Virtual Enterprise (WVE) and it became a part of WebSphere Network Deployment starting with version 8.5. IM introduces the On Demand Router which supports application editioning, health policies, service policies, maintenance mode, automatic discovery, dynamic clusters, traffic shaping, and more. The ODR was first delivered as a Java process that was based on the Proxy Server and it was normally placed in between a web server and the application servers. Starting with WAS 8.5.5, there is an option called ODRLib which is a native C component that delivers much of the same functionality but is integrated directly into the IBM HTTP Server (IHS) web server.
Java On Demand Router (ODR)

The Java On Demand Router (ODR) is built on top of the WAS Java Proxy Server. Both of these write the following log files asynchronously in a background "LoggerOffThread:"

- **local.log**: A log of the communications between the client (e.g. browser) and the ODR, i.e. the activities in the "local" ODR process.
- **proxy.log**: A log of the communications between the ODR and the backend server (e.g. application server).

The weighted least outstanding request (WLOR) load balancing algorithm is generally superior to the available load balancing algorithms in the WebSphere plugin. WLOR takes into account both the weight of the server and the number of outstanding requests, so it is better at evening out load if one server slows down. WLOR is the default in both ODRLib and the Java ODR.

The "excessive request timeout condition" and "excessive response time condition" are useful health policies that the ODR can monitor to gather diagnostics on anomalous requests.

Conditional Request Trace enables traces only for requests that match a particular condition such as a URI.

The ODR measures "service time" as the time the request was sent to the application server until the time the first response chunk arrives.


Maintenance Mode

Putting servers into maintenance mode is a great way to gather performance diagnostics while reducing the potential impact to customers. One maintenance mode option is to allow users with affinity to continue making requests while sending new requests to other servers.

Putting a server into maintenance mode is a persistent change. In other words, a server will remain in maintenance mode (even if the server is restarted) until the mode is explicitly changed. The maintenance mode of a server is stored persistently as a server custom property. The name of the custom property is "server.maintenancemode" under Application Servers > Administration > Custom Properties. Possible values for that property are:

- **false**: maintenance mode is disabled
- **affinity**: only route traffic with affinity to the server
- **break**: don't route any traffic to the server

Custom Logging

The Java ODR supports custom logging which logs information about HTTP responses, allows for conditions on what is logged and has very flexible fields for logging:

The condition uses HTTP request and response operands. Response operands include response code, target server, response time, and service time:
The fields available to print are:

Instructions to log all responses:

1. Log into the machine that runs the WAS DMGR, open a command prompt, and change directory to the ${WAS}/bin/ directory.
2. Run the following command for each ODR, replacing ${ODRNODE} with the ODR's node and %{ODRSERVER} with the name of the ODR:

   wsadmin -f manageODR.py insertCustomLogRule ${ODRNODE}:%{ODRSERVER} 1 "service.time >= 0" "http.log %h %t %r %s %b %Z %v %R %T"

3. In the WAS DMGR administrative console, for each ODR, go to: Servers > Server Types > On Demand Routers > ${ODR} > On Demand Router Properties > On Demand Router settings > Custom Properties
   1. Click New and set Name=http.log.maxSize and Value=100 and click OK. This value is in MB.
   2. Click New and set Name=http.log.history and Value=10 and click OK
   3. Click Review, check the box to synchronize, and click Save
4. Restart the ODRs
5. Now observe that there should be an http.log file in ${WAS}/profiles/${PROFILE}/logs/${ODR}/

The default value for http.log.maxSize is 500 MB and the default value for http.log.history is 1:

Note that the number of historical files is in addition to the current file, meaning that the defaults will produce up to 1GB in two files. Also note that changing the values affects not only the ODR custom logs, but also the proxy.log, local.log, and cache.log.

Other notes:

Log rules may be listed with:

   $ wsadmin -f manageODR.py listCustomLogRules ${ODRNODE}:%{ODRSERVER}
   WASX7209I: Connected to process "dmgr" on node dmgr1 using SOAP connector;
   The type of process is: DeploymentManager
   WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable:
   "[listCustomLogRules, odr1:odrserver1]"
   1: condition='service.time >= 0' value='http.log %h %t %r %s %b %Z %v %R %T'

Log rules may be removed by referencing the rule number (specified in insertCustomLogRule or listed on the left side of the output of listCustomLogRules):

   $ wsadmin -f manageODR.py removeCustomLogRule ${ODRNODE}:%{ODRSERVER} 1
   WASX7209I: Connected to process "dmgr" on node dmgr1 using SOAP connector;
   The type of process is: DeploymentManager
   WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable:
"[removeCustomLogRule, odr1:odrserver1, 1]"

Removed log rule #1

If the overhead of the example log rule above is too high, then it may be reduced significantly by only logging requests that take a long time. Change the server.time threshold (in milliseconds) to some large value. For example (the name of the log is also changed to be more meaningful such as http_slow.log):

```
$ ./wsadmin.sh -f manageODR.py insertCustomLogRule ${ODRNODE}:${ODRSERVER} 1
"service.time >= 5000" "http_slow.log %h %t %r %s %b %Z %v %R %T"
```

WASX7209I: Connected to process "dmgr" on node dmgr1 using SOAP connector;
The type of process is: DeploymentManager
WASX7303I: The following options are passed to the scripting environment and are available as arguments that are stored in the argv variable:
"[insertCustomLogRule, odr1:odrserver1, 1, service.time >= 5000, http_slow.log %h %t %r %s %b %Z %v %R %T]"
Inserted 'log rule #1

Example output:

```
localhost6.localdomain6 09/Jan/2018:14:33:55 PST "GET /swat/Sleep HTTP/1.1" 200 326 cell1/node1/dc1_node1 oc3466700346 6006 6004
```

Note that %r will be double-quoted without you needing to specify the double quotes in insertCustomLogRule. In fact, insertCustomLogRule does not support double quotes around any field.

**Binary Trace Facility (BTF)**

The Java ODR supports a different type of tracing from the traditional diagnostic trace. Btrace enables trace on a per-request basis and infrequently-occurring conditions out-of-the-box (e.g. reason for 503). Btrace is hierarchical with respect to function rather than code and trace records are organized top-down and left-to-right (processing order). The trace specification can be set as a cell custom property starting with trace, e.g. name=trace.http, value=http.request.loadBalance=2

The "trace" command in the WAS installation directory can be used to format btrace data:

```
${WAS}/bin/trace read ${SERVER_LOGS_DIRECTORY} ${SPEC_TO_READ}
```

**Dynamic clusters**

**Application Placement Controller (APC)**

The Application Placement Controller code runs in one JVM in the cell and coordinates stopping and starting JVMs when dynamic clusters are in automatic mode, or creating runtime tasks for doing so when dynamic clusters are in supervised mode. The frequency of changes is throttled by the minimum time between placements option (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/twve_odmonitora pc.html). Some of the basic theory of the APC is described here:

Investigate APC issues:

1. Check all node agents are running and healthy and the core group is marked as stable.
2. Check if any nodes or servers are in maintenance mode.
3. Check the logs for servers to see if they were attempted to be started but failed for some reason (e.g. application initialization).
4. Check each node's available physical memory if there is sufficient free space for additional servers.
5. Find where the APC is running (DCPC0001I/HAMI0023I) and not stopped (DCPC0002I/HAMI0023I), and ensure that it is actually running at the interval of minimum time between placement options (otherwise, it may be hung).
6. Check if APC detected a violation with the DCPC0309I message. If found, check for any subsequent errors or warnings.
7. Check the apcReplayer.log, find the "**BEGIN PLACEMENT INPUT DUMP**" section, and review if all nodes are registered with lines starting with {CI.

If APC is constantly stopping and starting JVMs seemingly needlessly, test various options such as:

- APC.BASE.PlaceConfig.DEMAND_DISTANCE_OVERALL=0.05
- APC.BASE.PlaceConfig.UTILITY_DISTANCE_PER_APPL=0.05
- APC.BASE.PlaceConfig.WANT_VIOLATION_SCORE=true
- APC.BASE.PlaceConfig.PRUNE_NO_HELP=false

**Service Policies**

Service policies define application goals (e.g. average response time less than 1 second) and relative priorities (e.g. application A is High). The Java ODR uses these policies in its request prioritization and routing decisions.

**CPU/Memory Overload Protection**

These overload protection features cause the Java ODR to queue work to application servers that it sees are over the configured thresholds of CPU and/or memory usage.

**Health Policies**

When using the "excessive memory usage" health policy (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/cw_ve_odhealth.html?lang=en), set usexdHeapModule=true (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rw_ve_odhealthcustprop.html?lang=en). Otherwise, the heap usage is sampled and this can create false positives with generational garbage collection policies such as gencon. The "memory leak" health policy uses the built-in traditional WAS performance advisor and this always samples, so it's not recommended with generational garbage collectors.

**Visualization Data Service**

This service logs key performance data into CSV log files. The logs are written to the deployment manager profile directory at $DMGR_PROFILE/logs/visualization/*.log
1. System Administration > Visualization Data Service > Check "Enable Log"
   1. Timestamp format = MM/dd/yyyy HH:mm:ss
      1. If this is not specified, it defaults to the "number of milliseconds since the
         standard base time known as "the epoch", namely January 1, 1970, 00:00:00
         GMT." – i.e. new Date(timestamp)
   2. Max file size = 20MB
   3. Max historical files = 5
      1. The max file size and historical files apply to each viz data log file, individually.

Example output of ServerStatsCache.log:
timeStamp,name,node,cellName,version,weight,cpu,usedMemory,uptime,totalRequests,liveSessions,up
dateTime,highMemMark,residentMemory,totalMemory,db_averageResponseTime,db_throughput,total
MethodCalls
01/03/2019 09:45:53,server1,localhostNode01,localhostCell01,XD
9.0.0.9,1,0.26649348143619733,80953,846,1337,0,01/03/2019 09:45:44,,334792,5137836,,

Bulletin Board over the Structured Overlay Network (BBSON)

BBSON is an alternative to the High Availability Manager (HAManager) and allows some of the WAS
components that traditionally relied on the HAManager to use a different approach. BBSON is built on
the P2P component which is peer-to-peer with small sized groups rather than a mesh network like
HAManager. This can allow for greater scalability and no need for core group bridges. All IM
components can use BBSON. WAS WLM can also use it: http://www-
01.ibm.com/support/docview.wss?uid=swg1PM71531

High Availability Deployment Manager (HADMGR)

The high availability deployment manager allows multiple instances of the deployment manager to
share the same configuration (using a networked filesystem) to eliminate a single point of failure if one
of them is not available: http://www-
01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/cw
ve_xdsodmgr.html. The HADMGR must be accessed through an On Demand Router (ODR) which
routes to one of the active deployment managers. The deployment manager can be very chatty in
making many small file I/O accesses, thus performance of the networked filesystem is critical.

Security

Use a clock synchronization service to keep system clock values as close as possible.
Security processing depends on time stamp validation and having clocks out of
synchronization more than five minutes can affect performance due to unnecessary re-
authentication and retry processing.
(http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/0710_largetopologies/LargeWebSphereTopologies.pdf)
Java Security

Java security typically reduces throughput by 15-40%. However, Java Security is not a fixed cost; rather, the cost is proportional to the number of security calls. One common manifestation of this is that one application has an overhead with Java Security enabled of X%, and then another application has a much higher overhead; in most cases, this is caused by a difference in the number of calls to security between those applications, rather than a product issue. A sampling profiler such as IBM Java Health Center is usually the best way to gauge the overhead of Java Security. Use the call stack invocation paths to reduce the number of security calls if possible.

Consider disabling Java 2 security manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk. ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tsec_tune.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tsec_tune.html))

Java 2 security has a significant performance cost, and therefore, do not use Java 2 security unless your application really requires it. ([http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/0710_largetopologies/LargeWebSphereTopologies.pdf](http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/0710_largetopologies/LargeWebSphereTopologies.pdf))

Single Sign On (SSO)

To configure SSO, click Security > Global security. Under Web security, click Single sign-on (SSO). SSO is only available when you configure LTPA as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 8.5 and does not support SSO. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. Some situations exist where SSO is not desirable and you do not want to use it in those situations.


Security Attribute Propagation


The following two custom properties might help to improve performance when security attribute propagation is enabled:

- com.ibm.CSI.propagateFirstCallerOnly: The default value of this property is true. When this custom property is set to true the first caller in the propagation token that
stays on the thread is logged when security attribute propagation is enabled. When this property is set to false, all of the caller switches are logged, which can affect performance.

- com.ibm.CSI.disablePropagationCallerList: When this custom property is set to true the ability to add a caller or host list in the propagation token is completely disabled. This function is beneficial when the caller or host list in the propagation token is not needed in the environment.


Horizontal Security Attribute Propagation

The SSO option will first check the local JVM's AuthCache. This is a "use based" cache in the Java heap of the JVM. A Subject used often can remain here until the LtpaToken expiration. Next, if security attribute propagation and Dynacache are enabled, WAS will check the ws/WSSecureMap DistributedMap. If the subject is not found here, WAS will try to make an MBean call back to the server that originally created the subject. The originating server's host:port is found in the SSO token. There is a timeout value that can be set to manage this condition:
com.ibm.websphere.security.tokenFromMBeanSoapTimeout. You can also disable the mbean callback:
com.ibm.websphere.security.disableGetTokenFromMBean. These properties were added at 7.0.0.17, PM28382.

When front-end servers are configured and in the same data replication service (DRS) replication domain, the application server automatically propagates the serialized information to all of the servers within the same domain [using ws/WSSecureMap].


The WSSecureMap security cache settings can be adjusted through custom properties in the administrative console.

- com.ibm.ws.security.WSSecureMapInitAtStartup=true
- com.ibm.ws.security.WSSecureMapSize (integer of 100 or greater).


In some cases, having [Web Inbound Security Attribute Propagation option on the Single sign-on (SSO) panel] enabled can improve performance. This improvement is most likely for higher volume cases where a considerable number of user registry calls reduces performance. In other cases, having the feature disabled can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.
You must determine whether enabling this option improves or degrades the performance of your system. While the option prevents some remote user registry calls, the deserialization and decryption of some tokens might impact performance. In some cases propagation is faster, especially if your user registry is the bottleneck of your topology. It is recommended that you measure the performance of your environment both by using and not using this option. When you test the performance, it is recommended that you test in the operating environment of the typical production environment with the typical number of unique users accessing the system simultaneously.

Note: Security attribute propagation may be set at multiple levels: cell, server, and security domain. For security domains, the option is set as a custom property with the name com.ibm.ws.security.webInboundPropagationEnabled and a value of true or false.

Explicit invalidations for ws/WSSecureMap are sent out on user logout. To disable this: com.ibm.websphere.security.web.removeCacheOnFormLogout=false

**LDAP Authentication**

Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.

- In the administration console, click Security > Global security.
- Under User account repository, click the Available realm definitions drop-down list, select Standalone LDAP registry and click Configure.
- Select the Ignore case for authorization option in the stand-alone LDAP registry configuration, when case-sensitivity is not important.
- Select the Reuse connection option.
- Use the cache features that your LDAP server supports.
- Choose either the IBM Tivoli® Directory Server or SecureWay directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.
- Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory can increase performance in group membership lookup. However, use Role only for group mechanisms.


### Secure Sockets Layer (SSL), Transport Layer Security (TLS)

- In some cases, using the unrestricted Java Cryptography Extension (JCE) policy file can improve performance. Refer to the information about tuning Web Services Security.

- Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.

- If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.

- Consider putting only an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use identity assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk (*) is used, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.


- When an SSL connection is established, an SSL handshake occurs. Once a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

- To enhance SSL performance, decrease the number of individual SSL connections and handshakes.

- Verify that the maximum number of keep alives are, at minimum, as large as the maximum number of requests per thread of the web server (or maximum number of processes for IBM® HTTP Server on UNIX). Make sure that the web server plug-in is capable of obtaining a keep alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection once a single request is processed. Also, the maximum number of threads in the web container thread pool should be larger than the maximum number of keep alives, to prevent the keep alive connections from consuming the web container threads.
Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in following 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.

The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware, for example, Data Encryption Standard (DES) and triple-strength DES (3DES); however, specialized hardware can provide efficient implementations of these same algorithms.

The Message Digest Algorithm (MD5) and Secure Hash Algorithm (SHA) are the two hash algorithms used to provide data integrity. MD5 is generally faster than SHA, however, SHA is more secure than MD5.

Data Encryption Standard (DES) and Rivest Cipher 2 (RC2) are slower than Rivest Cipher 4 (RC4). Triple DES is the most secure, but the performance cost is high when using only software.

The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.


Use certificates that are signed by a certificate authority (CA), preferably an internal CA for internal communications, whenever possible. This usage reduces the number of signers that are needed in a truststore and allows the replacement of a personal certificate without ramifications to clients.

You can use SSL offload devices to reduce the SSL overhead for internet and intranet facing applications. Using keepAlive, which is on by default, dramatically minimizes the SSL overhead, removing the SSL handshakes, which tends to be the largest overhead of SSL.

J2C Authentication Subjects

Read-only Subject enables a new cache for J2C Auth Subjects when using container-managed auth data aliases. If the J2C auth subject does not need to be modified after it is created, the following new tuning parameters can be used to improve Java 2 Security performance:

- com.ibm.websphere.security.auth.j2c.cacheReadOnlyAuthDataSubjects=true
- com.ibm.websphere.security.auth.j2c.readOnlyAuthDataSubjectCacheSize=50 (This is the maximum number of subjects in the hashtable of the cache. Once the cache reaches this size, some of the entries are purged. For better performance, this size should be equal to the number of unique subjects (cache based on uniqueness of user principal + auth data alias + managed connection factory instance) when role-based security and Java 2 security are used together).


Authentication Cache

Consider increasing the cache and token timeout if you feel your environment is secure enough. By increasing these values, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token timeout is the exposure of having a token hacked and providing the hacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms. (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tsec_tune.html)

CSIv2 Cache

Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.

Consider changing the values for the CSIv2 session cache. Changing these values can avoid resource shortages. Refer to the Common Secure Interoperability Version 2 outbound communications topic for more information.

If you are communicating only with WebSphere Application Server Version 5 or higher servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

UserAuthorization

Map your users to groups in the user registry. Associate the groups with your Java Platform, Enterprise Edition (Java EE) roles. This association greatly improves performance when the number of users increases.


ServletSecurityConstraints

Judiciously assign security-constraints for servlets. For example, you can use the * .jsp URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the * .jsp, * .do, * .html extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.


EJBMethodPermissions

Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.


AdministrativeSecurity

Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.


If WebSphere Application Server security is used only to protect administrative access, disable application security so that the collaborators do not perform actions that might affect throughput.

(http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/0710_largetopologies/LargeWebSphereTopologies.pdf)
Expired Certificates

1. On the SSL certificate and key management page, click the Keystores and Certificates link in the Related Items list.
2. Click the Keystore,
3. Click Personal certificate > Additional Properties.
4. Renew the certificate and exchange the signer certificates between the DMGR and Nodes:
   1. Go to SSL certificate and key management > Manage endpoint security configurations > Click on inbound on the node (NodeDefaultSSLSettings,null) > click on Key stores and certificates
   2. Select both CellDefaultKeyStore and CellDefaultTrustStore by checking the box and click on exchange signers under CellDefaultKeyStore personal certificates. Choose all certificates and click add click add and it will add all those certificate under CellDefaultTrustStore signers and then click OK
   3. Same thing but select both CellDefaultKeyStore and NodeDefaultTrustStore by checking the box and click on exchange signers under CellDefaultKeyStore personal certificates. Choose all certificates and click add and it will add all those certificate under NodeDefaultTrustStore signers and then click OK
   4. Same thing but select NodeDefaultKeyStore and CellDefaultTrustStore by checking the box and click on exchange signers under NodeDefaultKeyStore personal certificates. Choose all certificates and click add and it will add all those certificate under CellDefaultTrustStore signers and then click OK
   5. Same thing but select NodeDefaultKeyStore and NodeDefaultTrustStore by checking the box and click on exchange signers under NodeDefaultKeyStore personal certificates. Choose all certificates and click add and it will add all those certificate under NodeDefaultTrustStore signers and then click OK
5. Save the changes with the master configuration and restart the dmgr.
6. Stop the nodeagent and JVMs and manually sync the node with dmgr using syncNode
   1. If the syncNode fails (e.g. ADMU0127E), then manually copy the key.p12 file from the node's configuration in the DMGR profile to the node's configuration.
7. Start the nodeagent and see the status in the admin console and sync the node from the console and see if sync is going smooth by tailing the nodeagent logs.

Administration

Deployment Manager

The memory requirement of the deployment manager increases as the size of the topology increases, and as the number of concurrent sessions increases. Since the deployment manager is just a single process, there is no mechanism to balance the load. Therefore, there is a limit to the number of concurrent users that can be supported on a single deployment manager.
Just as you would tune the application server heap size, you need to tune the deployment manager heap size to accommodate the number of concurrent users who access the deployment manager. Enable verbose garbage collection, and observe how the heap size increases with the increase in topology and in the number of users.

If too many concurrent sessions are overloading the deployment manager, you need to place a limit on concurrent access. For scripting, consider using the V7 job manager as a mechanism for users to submit wsadmin jobs. The jobs are run sequentially, and an email notification is sent to the user upon job completion.

A JMX request from the deployment manager to a single application server flows through the deployment manager to the node agent on the same node where the server resides, and finally to the application server itself. This design is intended for scalability. The deployment manager has to communicate with a node agent only, and each node agent has to communicate with its respective application servers only.

If an invocation is made to all of the servers on a node, the deployment manager uses one invocation to the node agent and the node agent, in turn, broadcasts the invocation to every server on the node. To avoid a scenario where queries get stuck, use narrow queries that target only the servers or nodes from which you really need information. Queries that touch every server can considerably consume cell resources.

Use -Dcom.ibm.ws.management.connector.soap.keepAlive=true to avoid the cost of SSL re-handshaking when AdminClient uses PullRemoteReceiver/PullRemoteSender.

Starting with WAS 8.5.5.7 (PI42208), you may set -Dcom.ibm.console.overrideSyncPref=true on the deployment manager so that saving any changes will automatically synchronize with any running nodes. This avoids common issues with junior administrators that save a change and restart a server before the automatic synchronization kicks in.

**wsadmin/JMX**

Often in a script you need to search for a specific configuration object, such as a specific node, server, or data source. The configuration service extracts what you are searching from the master repository to the workspace for you to make your changes. How you construct your query can greatly affect how many files are extracted. If you do not use a targeted query, you can potentially cause the entire repository to be extracted. For a large topology this is a very expensive operation.

Starting the wsadmin process may take 20 seconds or more, depending on hardware. Avoid breaking up your configuration operations into multiple wsadmin invocations. Do combine them into a single script that can be run within one wsadmin session. Consider structuring your scripts into multiple files, and import them from a front-end script.
The -conntype NONE option is running wsadmin in local mode. We don't support updating the configuration in local mode while the deployment manager is running. After the change is made, to reflect to the changes to the nodes, the user will need to start the dmgr and run the node sync (syncNode) operation in order to sync the changes to the nodes. In local mode, the user will not be able to run anything operational such as AdminControl commands to invoke any WAS MBeans (and some of the AdminTask commands also require that the server is running). Other than that, local mode should act the same.

Getting diagnostics:

- AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "dumpThreads")
- AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "generateHeapDump")
- AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "generateSystemDump")

Useful primers on Jython/wsadmin: [http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a123a55117b6ad7e3862572d5001834b6/$FILE/WP101014-%20-WSADMIN%20zOS%20V61%20Primer%20with%20Jython.pdf](http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a123a55117b6ad7e3862572d5001834b6/$FILE/WP101014-%20-WSADMIN%20zOS%20V61%20Primer%20with%20Jython.pdf) and [http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a392b0b65a0e9ff86862572670063f7e/$FILE/WP100963-%20-Jython%20Scripting%20tutorial.pdf](http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a392b0b65a0e9ff86862572670063f7e/$FILE/WP100963-%20-Jython%20Scripting%20tutorial.pdf)

Examples

Restart server:

```
print "Restarting " + sys.argv[0] + "/" + sys.argv[1] + "..."
print
AdminControl.invoke(AdminControl.queryNames("WebSphere:*",type=Server,node=" +
sys.argv[0] + ",process=" + sys.argv[1]), "restart")
print "Restart asynchronously started..."
```

The only potential problem with the above is that it fires off the restart asynchronously, so you don't know if it succeeded or not. Instead, the script can be changed to invoke a stop and then a start, the first of which is synchronous and reports any errors:

```
print "Stopping " + sys.argv[0] + "/" + sys.argv[1] + "..."
print AdminControl.stopServer(sys.argv[1], sys.argv[0])
print "Starting " + sys.argv[0] + "/" + sys.argv[1] + "..."
print AdminControl.startServer(sys.argv[1], sys.argv[0])
print "Done"
```

Querying PMI

```
# Provide the name of the WebSphere Application Server
```
serverName = "server1"

# Example PMI object:
https://www.ibm.com/support/knowledgecenter/SSAW57_9.0.0/com.ibm.websphere.nd
.multiproduct.doc/ae/rprf_datacounter4.html
pmiObject = "JVM"

# If serverName is not unique across the cell, add "node=N," before
"process":
lookup = "process=" + serverName

objectName = AdminControl.completeObjectName("type=Perf," + lookup + ",,*")
if objectName == '' or objectName is None:
    print "Server not running or not found"
else:
    # Query PMI:
    stats = AdminControl.invoke_jmx(AdminControl.makeObjectName(objectName),
"getStatsObject",
[AdminControl.makeObjectName(AdminControl.completeObjectName("type=" +
 pmiObject + ",," + lookup + ",,*")), java.lang.Boolean("false")],
["javax.managementObjectName","java.lang.Boolean"])

    # See the various "All Known Subinterfaces" for available methods
    depending on the statistic type (in this example, a CountStatistic):
    #

usedmem = stats.getStatistic("UsedMemory").getCount()
totalmem = stats.getStatistic("HeapSize").getCurrent()
percentUsed = int((float(usedmem)/float(totalmem))*100.0)

    print("Used Java Heap (MB): %s" %(usedmem/1024))
    print("Current Java Heap Size (MB): %s" %(totalmem/1024))
    print("Percent Java Heap Used: %s" %(percentUsed))

Node Synchronization

By default is set to 1 minute: http://www-
01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiproduct.doc/ae/ua
gt_synchservice.html. This can be increased to 60 minutes. Do not disable Automatic Synchronization
as it can affect security components such as LTPA key distribution.

Node synchronization is the process by which the WebSphere configuration is transferred
from the deployment manager to the node agent. The deployment manager and node agents
compare MD5 hashes of the configuration files to determine whether the files are identical.
In the cases of a node agent or deployment manager restart, the respective server must
create all the MD5 hashes in memory for all the configuration documents in the node or
cell. As the cell size and number of documents become larger, the start-up time also
increases.

WebSphere Application Server has added support for “Hot Restart Sync.” With this
support, the node agent and deployment managers save the hashes in both memory as well
as on the file system. When a restart is performed, the MD5 hashes do not need to be recomputed but rather can be loaded directly from disk. To enable this support, add the following custom property to your deployment manager and node agent:
-DhotRestartSync=true

Notifications
The SOAP connector has the advantage of having a better chance of making it through a firewall (since it is HTTP traffic) than RMI/IIOP; however, you will generally receive notifications faster with RMI than with SOAP. This is because the RMI uses a "push" model while SOAP uses a "pull" model.

When the RMI connector is used, a remote object is created on the client side and on the stub passed to the server side. Whenever a notification is received on the server, it is almost immediately sent (or "pushed") to the client and handed to the registered listeners. With SOAP, at regular intervals, the client requests any notifications from the server for this listener. If there are any, they are returned from (or "pulled" from) the server and then handed to the listeners. This occurs approximately every 20 seconds, but can be more frequent if a large number of notifications are being received.

Since notifications can take up to 20 seconds to be received when using the SOAP connector, it is recommended that the RMI connector be used to receive notifications, when possible.

Session Initiation Protocol (SIP)
UDP and Linux tuning:

Consider JVM and thread pool tuning:

WAS Traditional on z/OS
This chapter outlines any z/OS specific performance notes for WAS. See the z/OS operating systems chapter for prerequisite knowledge. Unless otherwise noted, everything covered in the general WAS section still applies.

Keep the number of nodes per local partition (LPAR) between one or two nodes with a maximum of four nodes per LPAR. Spread a cell or cluster over at least two LPARs. Using multiple LPARs ensures hardware redundancy as well, while still allowing the cluster to be upgraded on a per node basis.

http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/0710_largetopologies/LargeWebSphereTopologies.pdf
IBM recommends that you install as much of the WebSphere Application Server for z/OS code in the Link Pack Area (LPA) as is reasonable. Also, ensure that you have eliminated any unnecessary STEPLIBs which can affect performance. If you must use STEPLIBs, verify that any STEPLIB DDs in the controller and servant procs do not point to any unnecessary libraries.

The first place to review is your CTRACE configuration. Ensure that all components are either set to MIN or OFF. To display the CTRACE options for all components on your system, issue the following command from the operator console: D TRACE,COMP=ALL

To change the setting for an individual component to its minimum tracing value, use the following command, where xxx is the component ID: TRACE CT,OFF,COMP=xxx

This configuration change eliminates the unnecessary overhead of collecting trace information that is not needed. Often during debug, CTRACE is turned on for a component and not shut off when the problem is resolved.

Ensure that you are not collecting more SMF data than you need. Review the SMFPRMxx settings to ensure that only the minimum number of records is collected.

The Transaction Service RLS_LOGSTREAM_COMPRESS_INTERVAL custom property can be set to a value larger than the default value if the Transaction Service is the only application component using a logstream. If none of your components are configured to use a logstream, you can set this property to 0 (zero) to disable this function.

If you find long garbage collection pause times but the normal components of a pause (mark, sweep, compact, exclusiveaccess) do not add up to the total time, then this is usually caused by the Virtual Lookaside Facility (VLF) caching being disabled or not working efficiently.

"Ensure that ras_trace_defaultTracingLevel=0 or 1, and that ras_trace_basic and ras_trace_detail are not set."

Address Spaces

Each application server is split into two or more address spaces: a control region and one or more servant regions. The control region handles incoming traffic and distributes it to the servant regions where the application work is performed. It is a best practice to use ${X} as the control region name and ${X}S for the servant region names. For example, WBESR12 and WBESR12S.
WebSphere allows you to configure a minimum and maximum number of servants for a server. WLM will dynamically adjust the number of servants within the specified range, up or down based on what’s needed to meet the goals for the system. WLM does this for work running in WebSphere and for work elsewhere on the system.

To set the minimum value, consider how many servants you want to start automatically when the server is started and how many you want WLM to keep available. In determining the maximum value, consider how many servants you can support on your system. Also, consider the number of available connectors for applications in WebSphere and elsewhere in the system.

But what if something changes someday and the minimum just is not enough? Or, you reach the configured maximum and need more servants? To change the values, you must update the configuration and recycle the server. But if you are running at peak utilization and decide you need to increase the maximum number of servants; recycling the whole server is probably going to hurt more than just not having enough servants. It would be nice to be able to dynamically change the number of servants without a recycle.

In Version 7, we introduced a new MODIFY command to let you do that. If the server is not configured as single-servant, you can change the current minimum and maximum number of servants. You enter the command as follows:

```
MODIFY server,WLM_MIN_MAX=(minimum,maximum)
```

Specify these values as decimal numbers. Obviously, the minimum must be less than the maximum.

Your changes are in effect until the next time you recycle the server, in which case, the
values in the configuration are used instead. To make your changes permanent, you need to update the configuration.

In general, WLM responds quickly to your request. If the minimum number of servants is not already running, WLM starts more. Increasing the maximum value, however, might not have any immediate effect. Further, decreases in values might also not cause an immediate change because of WLM’s opinion as to how many servants it needs. Some situations, such as session data pinned to the servant, might prevent WLM from reducing the number of currently active servants. Of course, unless you’ve committed your min and max values to memory, you would probably like to have a quick way to see what you are currently configured for. We added a new command to allow you to do that.

MODIFY server,DISPLAY,WLM

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/da939fa8cdf48510862575a1007461d2/$FILE/WP101464%20-%20WebSphere%20zOS%20Hidden%20Gems2.pdf

Start servants in parallel: wlm_servant_start_parallel=1 (http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/da939fa8cdf48510862575a1007461d2/$FILE/WP101464%20-%20WebSphere%20zOS%20Hidden%20Gems2.pdf)

All of the various custom properties, environment variables, etc. that are set in the WAS configuration (e.g. through the admin console) ultimately get generated into the was.env file (located under profiles/default/config/cells/cellname/nodes/nodename/servers/servername/was.env) which is read when the address space starts.

**Control Region**

The default value of worker threads in a control region is 25. This can be changed to a higher value as required by setting customer property was.controlThreads as follows:

Application servers > server_name > Container Services > ORB Service > Custom Properties > was.controlThreads

To verify how many control region threads you are using, you can check the following message in the control region joblog:

```
BBOM0001I control_region_thread_pool_size: 25.
```

Starting in WAS 8.0.0.8 and 8.5.5.2 (PM85194), use the property control_region_thread_pool_maximum_size to allow growth of this pool, or set to 0 to allow dynamic calculation of the size (see also P150098).

**Daemon**

"Stopping a Daemon server will stop all servers for that cell on that... image. This is because of the way servers for that cell... access key LPA modules. It's done "through" the Daemon server. Stopping the Daemon server means the servers can no longer access those modules, so they too stop." (http://www-
Thread Pools

Most work in the servant is handled by the ORB thread pool. The maximum size of this pool is controlled by the ORB workload profile setting:


- **IOBOUND**: Default - Number of threads is 3 * Number of processors. Specifies more threads in applications that perform I/O-intensive processing on the z/OS operating system. The calculation of the thread number is based on the number of processors. IOBOUND is used by most applications that have a balance of processor intensive and remote operation calls. A batch job is an example that uses the IOBOUND profile.

- **CPUBOUND**: Number of threads is the number of processors. Specifies that the application performs processor-intensive operations on the z/OS operating system, and therefore, would not benefit from more threads than the number of processors. The calculation of the thread number is based on the number of processors. Use the CPUBOUND profile setting in processor intensive applications, like compute-intensive (CI) jobs, XML parsing, and XML document construction, where the vast majority of the application response time is spent using the processor.

- **LONGWAIT**: Number of threads is 40. Specifies more threads than IOBOUND for application processing. LONGWAIT spends most of its time waiting for network or remote operations to complete. Use this setting when the application makes frequent calls to another application system, like Customer Information Control System (CICS®) screen scraper applications, but does not do much of its own processing.

- In WebSphere Application Server for z/OS V7 you can choose Workload profile CUSTOM and then set property servant_region_custom_thread_count to the number of servant threads you want up to a limit of 100.

BBOO0234I SERVANT PROCESS THREAD COUNT IS X

WAS 7 on z/OS introduced the ability to interrupt hung threads: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/3c02b79e79ea32fd8625751a005d7f63/$FILE/WP101374%20-%20WebSphere%20zOS%20V7%20Dispatch%20Timeout%20Improvements.pdf. This was improved in WAS 8: http://w3-03.ibm.com/support/techdocs/atsmastr.nsf/3af3af29ce1f19cf86256c7100727a9f/d7bb7aa1f7be24128625791e00830412/$FILE/WP101992%20-%20WebSphere%20zOS%20V8%20Hidden%20Gems.pdf

Joblogs

Type ? next to the WAS servant region in the SDSF.DA or SDFS.ST panels. Roughly speaking, SYSPRINT is equivalent to SystemOut.log and SYSOUT is equivalent to SystemErr.log + native_stderr.log

Common things to look for in WAS joblogs:

- Search for the word HOST by typing F HOST and F5 to repeat search
  - Hostname: com.ibm.CORBA.LocalHost = ZTESTB2.PDL.POK.IBM.COM
- Search for the word LEVEL by typing F LEVEL and F5 to repeat search
• WAS Level: BBOM0007I CURRENT CB SERVICE LEVEL IS build level 6.1.0.32 (AM24112) release WAS61.ZNATV date 10/10/10 19:40:16.
• Search for the word cell_name
  • Cell name: cell_name: wbecell.
• Search for the word PROCEDURE by typing F PROCEDURE and F5 to repeat
  • PROCLIB: PROCEDURE WBESS62 WAS EXPANDED USING SYSTEM LIBRARY USER.S12.PROCLIB
• Search for the word WAS_HOMe by typing F WAS_HOME and F5 to repeat

**Timeouts**

See [http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/3c02b79e79ea32fd8625751a005d7f63/$FILE/WP101374-%20-%20WebSphere%20%20zOS%20%20V7%20Dispatch%20Timeout%20Improvements.pdf](http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/3c02b79e79ea32fd8625751a005d7f63/$FILE/WP101374-%20-%20WebSphere%20%20zOS%20%20V7%20Dispatch%20Timeout%20Improvements.pdf)

**WLM**

WebSphere has several different types of work running in its address spaces. It is classified using classification rules under 3 different workloads:

1. For STC workloads the WebSphere address spaces control regions and servant regions would be given an aggressive Velocity goal equal to or slightly less than DB2, IMS, or MQ and a goal equal to or slightly higher than CICS.
2. For OMVS workloads the WebSphere address spaces control regions and servant regions would be given an aggressive Velocity goal so that at start-up the BPXBATCH facility used to run our applyPTF.sh script does not slow startup of the server.
3. For CB workloads the WebSphere Servant Regions are given a Response time with percentile goal close to but not to exceed 90% of the work in .5 seconds. Even though WebSphere servers are long running tasks, typically Velocity goals are used for long running tasks, the actual transactions within WebSphere are very short lived HTTP type transactions. Response times with percentile goals are used for these short lived transactions.

The report classes associated with the classification rule for each workload would be unique.

Workload CB is enclave work or WLM queue managed WebSphere work. Almost all WebSphere work happens here after the initial startup of the address spaces.

STC work also occurs in the WebSphere address spaces:

• The processing necessary to start the address spaces before the first enclave is created is STC workload.
• Any spawned threads from the application will not be enclave, WLM, or CB managed work and will run under STC.
• Address space functions such as JES related activities will be STC workload.
• An argument can be made that says Garbage Collection activities run under STC workload.
Rarely use discretionary classification for WebSphere.

OMVS work also occurs in the WebSphere Address Spaces. During startup a program called BPXBATCH is executed to run a script in the JCL. This script called applyPTF.sh checks to see if any service has been applied. If service has been applied this script executes any post install actions necessary. If startup is very slow, you may want to investigate a classification rule for OMVS. If the combination of applyPTF.sh and lack of classification are the cause of the slow startup, adding a classification rule may fix the slow start problem.

More on WAS with WLM: http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/f1ec690b6bee04cd8625778800595c69/$FILE/WP101740-%20-%20WAS_and_zOS_WLM_v8.pdf

WebSphere creates a WLM enclave for all requests that get dispatched in a servant. An enclave has an associated WLM service class and report class. The service class is used by WLM to help make decisions about assigning resources to the servant to ensure the requests meet the goals defined in the service class. The report class is used to separate information about the requests in reports generated by RMF (or other similar products). To determine which service class and report class to assign to the enclave, WLM looks at classification information provided by WebSphere when the enclave is created.

One piece of information provided is called a transaction class. This is just an eight character name assigned to the request. WebSphere supports an XML file pointed to by the variable wlm_classification_file to determine what transaction class to use. The XML file allows you to specify a different transaction class (and thus indirectly a different service class and report class) for different applications or even parts of applications.

The XML file also allows you to specify a transaction class to be used when classifying requests that are internally generated. Sometimes the controller needs to dispatch something in its own servant. For example, this could be the dispatch of a management bean (MBean). To separate internal work from application work, you might want to put these requests in their own report class. To do that you simply specify the ‘internal’ clause in the XML file and provide a transaction class name that WLM will recognize (based on the rules you provide in the WLM configuration) and assign service and report classes appropriately.

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/da939fa8cdf48510862575a1007461d2/$FILE/WP101464-%20-%20WebSphere-%20zOS-%20Hidden%20Gems2.pdf

SMF 120
Details on 120-9 records in WP-101342

MODIFY Command
/F ADDRESSSPACE,... COMMANDS... or /MODIFY ADDRESSSSPACE,... COMMANDS (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.}

Page 353
Display Java heap information: /F ADDRESSSPACE,JVMHEAP

Generate a javacore: /F ADDRESSSPACE,JA VACORE

The joblog will show where it is written: JVDUMP007I JVM Requesting Java Dump using '/var/WebSphere/home/ZPSRG/javacore.20090309.205027.50397255.txt

In version 8 we added an option to these commands to specify the ASID of the servant region you want dumped. Just add a “ASIDX=” after the command with the appropriate ASID (in hex) of the servant region you want to dump. For example (http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/2e8a87fddeb bb3328625791e0083041f/$FILE/WP101992%20-%20WebSphere%20zOS%20V8%20Hidden %20Gems.pdf):

/F server,JA VACORE,ASIDX=F4

PAUSELISTENERS, will cause the target server to close its listener ports and stop taking new requests

DISPLAY,SERVERS modify command has been enhanced to also report the 'state' of the server. There are four possibilities: ACTIVE, ENDING, PAUSED/STOPPING, and RECOVERY. ACTIVE seems pretty obvious. Basically ACTIVE means it isn't any of the other states; it could be up or it could be initializing. ENDING means that the server is on its way down. PAUSED/STOPPING means either you have issued PAUSELISTENERS or STOPped the server. It is kind of the same thing. In both cases the server is not taking new work, but there is a possibility work is still in-flight inside the server. The only difference is if we are stopping, then once the work completes the server will end. Finally, RECOVERY means that the server has been started to recover in-flight transactions and will automatically shut down once that is done. No new work will be taken.

BBOO0182I SERVER ASID SYSTEM LEVEL STATE
BBOO0183I WAS00 /ZWASAXXX 6Fx SY1 8.0.0.0 (ff1106.32) ACTIVE
BBOO0183I BBON001 /BBON001 58x SY1 8.0.0.0 (ff1106.32) ACTIVE
BBOO0183I BBCO001 /BBOS001 5Bx SY1 8.0.0.0 (ff1106.32) PAUSED/STOPPING
BBOO0183I BBODMGR /BBODMGR 57x SY1 8.0.0.0 (ff1106.32) ACTIVE

Way back in WebSphere Version 5 we introduced the DISPLAY,WORK command (underneath the MVS 'Modify' command for the WAS controller). This pretty cool command lets you see how much work had been processed by the server since it started and how much work was actually in the server at the time. You could even look server region by server region and see how work was spreading (or not) across them. (http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/ec31a38f42fa f8c486257c27005f7a64/$FILE/WP102371%20-%20New%20Functions%20in%20WAS %20zOS%20Maintenance%20Stream.pdf)

WebSphere Application Server Liberty

Liberty product documentation: https://www.ibm.com/support/knowledgecenter/SSAW57_liberty

Liberty PoC USB Edition:
A subset of Liberty is open source: https://github.com/OpenLiberty/open-liberty

**WAS Liberty Recipe**

In addition to the overall recipe in the WebSphere Application Server chapter,

1. Liberty has a primary thread pool instead of multiple thread pools in the WAS Traditional Profile and this is auto-tuned based on throughput. Set executor maxThreads for manual control.
2. Database connection pools should not be consistently saturated: <connectionManager maxPoolSize="X" ... /></.
3. Monitor response times of key application components (e.g. servlets, databases, MDBs, etc.).
4. Set the maximum keep alive requests to unlimited: <httpOptions maxKeepAliveRequests="-1" />
5. Review logs for any errors, warnings, or high volumes of messages.
7. Enable the monitor feature and related connector to have the ability to dynamically monitor metrics of a running JVM.
8. In production environments, disable the application auto-update monitoring capabilities.
10. When using messaging, review the maxBatchSize and maxConcurrency of JMS activation specification configurations.
11. Consider enabling the HTTP access log with response times.
12. If there is a performance problem, test with <execututor coreThreads="X" maxThreads="X" /></ where X = NUM_CPUs*2. If this is better, consider opening a support case to investigate why the default is not optimal.

**server.xml**

Available configuration options in the most recent OpenLiberty are here: [https://openliberty.io/docs/ref/config/](https://openliberty.io/docs/ref/config/)

**jvm.options**

Generic JVM arguments are set either in ${WLP}/usr/servers/${SERVER}/jvm.options for a particular server or in ${WLP}/etc/jvm.options as defaults for all servers. Put each option on its own line in jvm.options. [http://www-01.ibm.com/support/docview.wss?uid=swg21596474](http://www-01.ibm.com/support/docview.wss?uid=swg21596474)
Verbose Garbage Collection

As outlined in the Java chapters, enabling verbosegc is recommended. This can be enabled in the jvm.options file (see above). For example, on the IBM JVM, add -Xverbosegclog:verbosegc.%Y%m%d.%H%M%S.%pid.log,5,51200

Logs and Trace

Always provide or analyze both console.log and messages.log as they may have different sets of messages. Ideally, when gathering Liberty server logs, gather the entire logs folder under $WLP/usr/servers/${SERVER}/logs, as well as the server.xml under $WLP/ust/servers/$ {SERVER}/

Traditional WAS XCT and Request Metrics are not currently available in Liberty.


Liberty uses a Java retransformation agent to supports some of its logging capabilities. On IBM Java < 7.1, the mere presence of a retransformation agent will cause the VM to double the amount of class native memory used, whether any individual class is transformed or not. On IBM Java >= 7.1, additional memory is allocated on-demand only for the classes that are retransformed.

**console.log**

console.log includes stdout, stderr, WAS messages (except trace) equal to or above the threshold set by consoleLogLevel (by default, AUDIT), and System.out plus System.err (if copySystemStreams is true, which it is by default) without timestamps. The console.log is always truncated on server startup and does not support maximum size nor rollover. For those experienced with traditional WAS, by default, console.log is like the combination of native_stdout.log, native_stderr.log, System.out, System.err, and WAS messages above AUDIT in SystemOut.log.

If you would like to use console.log for stdout and stderr and use messages.log for everything else (note: a couple of AUDIT messages will still show up before the logging configuration is read):

```xml
<logging copySystemStreams="false" consoleLogLevel="OFF" />
```

**messages.log**

messages.log includes WAS messages (except trace) equal to or above the INFO threshold (non-configurable), java.util.logging.Logger messages equal to or above traceSpecification (default *=info), System.out, and System.err, with timestamps. The old messages.log is always rotated on server startup into a historical file. The messages.log supports maximum size and rollover. For those experienced with traditional WAS, messages.log is like the combination of SystemOut.log and SystemErr.log.

**trace.log**

Diagnostic trace is enabled with traceSpecification ([http://www-01.ibm.com/support/docview.wss?uid=swg21596714](http://www-01.ibm.com/support/docview.wss?uid=swg21596714)). For example:
If trace is enabled, trace.log includes WAS messages, java.util.logging.Logger messages, WAS diagnostic trace, System.out, and System.err, with timestamps.

**HTTP Access Logs**

HTTP access logging with response times may be used to track the number and response times of HTTP(S) requests:


APARs PI20149 and PI34161 are required for accurate response times, which are available in Liberty 8.5.5.6: [http://www-01.ibm.com/support/docview.wss?uid=swg1PI20149](http://www-01.ibm.com/support/docview.wss?uid=swg1PI20149) and [http://www-01.ibm.com/support/docview.wss?uid=swg1PI34161](http://www-01.ibm.com/support/docview.wss?uid=swg1PI34161)

For example, use the following logFormat in server.xml:

```xml
<httpEndpoint ...>
  <accessLogging filepath="${server.output.dir}/logs/access.log" maxFileSize="250" maxFiles="4" logFormat="%h %i %u %t "%r" %s %b %D" />
</httpEndpoint>
```

Available options for logFormat:


See the [WAS Traditional Profile HTTP chapter](http://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rrun_chain_httpcustom.html) for details on how to review and graph this output.

**Binary Logging**

Binary Logging is essentially the same as HPEL logging in traditional WAS with the same benefits:


1. Add the following additional line to bootstrap.properties for the server (if no such file exists, create the file in the same directory as server.xml):

   ```properties
   websphere.log.provider=binaryLogging-1.0
   ```

2. Configure binaryLogging in server.xml by replacing any existing `<logging>` element with:

   ```xml
   <logging consoleLogLevel="OFF" traceSpecification="*=info">
     <binaryLog purgeMaxSize="1024" />
     <binaryTrace purgeMaxSize="1024" />
   </logging>
   ```

**Request Timing**

Request timing gives the duration of requests along with a dump of what events have occurred during
the request (e.g. which servlets run, which JDBC requests are made, etc.), and it would give a stack trace. It's configurable so that it can be set to do these things only when the duration of the request exceeds a specified threshold (e.g. 30 seconds). It also has a unique requestID. http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/aewlp_requesttiming.html

```
<featureManager>
  <feature>requestTiming-1.0</feature>
</featureManager>

<requestTiming slowRequestThreshold="5s" hungRequestThreshold="60s"
includeContextInfo="true" sampleRate="1" />
```

Example output:

```
com.ibm.ws.request.timing.manager.SlowRequestManager         W TRAS0112W:
Request AAAAqqZnfKN_AAAAAAAAAAB has
been running on thread 000000c2 for at least 5000.936ms. The following stack trace shows what this thread is currently running.
   at java.lang.Thread.sleep(Native Method)
   at java.lang.Thread.sleep(Thread.java:922)
   at com.ibm.pd.Sleep.service(Sleep.java:25)
   at javax.servlet.http.HttpServlet.service(HttpServlet.java:668)...
```

The following table shows the events that have run during this request.

<table>
<thead>
<tr>
<th>Duration</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>5003.810ms</td>
<td>websphere.servlet.service</td>
</tr>
</tbody>
</table>

**Event Logging**

Event logging is an easy way to log events such as HTTP responses and durations: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/aewlp_feature_eventLogging-1.0.html?lang=en

```
<featureManager>
  <feature>eventLogging-1.0</feature>
</featureManager>

<eventLogging includeTypes="websphere.servlet.service" minDuration="500ms"
logMode="exit" sampleRate="1" includeContextInfo="true" />
```

Example output:

```
[10/1/15 14:10:57:962 UTC] 00000053 EventLogging I END
requestID=AAABqGA0rs2_AAAAAAAAAAA # eventType=websphere.servlet.service #
contextInfo=pd | com.ibm.pd.Sleep?durationms=10000 # duration=10008.947ms
```

For the reason why you might want to enable event logging, consider the following case: You've set the requestTiming threshold to 10 seconds which will print a tree of events for any request taking more than 10 seconds. However, what if a request occurs which has three database queries of 1 second, 2 seconds, and 6 seconds. In this case, the total response time is 9 seconds, but the one query that took 6 seconds is presumably concerning, so event logging can granularly monitor for such events.
Start-up

Liberty profile has a fixed timeout of 30 seconds for applications to start. After the 30
second timeout expires two things happen: a message is output to the logs saying the
application didn't start quickly enough; during server startup the server will stop waiting for
the application to start and claim to be started, even though the application is not yet ready.

To set this value to [for example] one minute add the following to server.xml:

<applicationManager startTimeout="1m"/>

https://www-01.ibm.com/support/docview.wss?uid=swg1PI51375

Collectives and High Availability

"The set of Liberty servers in a single management domain is called a "collective." A collective consists
of at least one server with the collectiveController-1.0 feature enabled that is called a “collective
controller.” Optionally, a collective can have many servers with the collectiveMember-1.0 feature
enabled that are called "collective members" and a collective can be configured to have many collective
controllers." (http://www-
01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/tagt_wlp_c
onfigure_collective.html?lang=en)

"You can organize Liberty servers into collectives to support clustering, administration, and other
operations that act on multiple Liberty servers at a time in order to efficiently and accurately deliver
application services to your organization." (http://www-
01.ibm.com/support/knowledgecenter/SSD28V_8.5.5/com.ibm.websphere.wlp.core.doc/ae/tagt_wlp_c
onfigure_collective.html?lang=en)

Collectives are a replacement for the WAS Classic High Availability Manager and core groups/cells.

Auto Scaling

"The Auto Scaling for Liberty feature provides an autonomic scaling capability of Liberty servers. Auto
Scaling can dynamically adjust the number of Java virtual machines (JVMs) that are based on
workload and dynamic routing to intelligently manage your workload. This feature provides
operational agility and decreases administrative overhead to enhance the resiliency of your middleware
environment. The conditions for auto scaling are defined by scaling policies. These conditions include,
the minimum/maximum number of server instances and the threshold values for each of the server
resources." (http://www-
01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/a
e/twlp_wve_autoscaling.html)

Thread Pools

Unlike traditional WAS which has many thread pools, most work in Liberty occurs in a single thread
pool named "Default Executor." The <executor /> element in server.xml may be used to configure the
Default Executor; although, in general, unless there are observed problems with threading, it is not recommended to tune nor even specify this element. The coreThreads attribute specifies the minimum number of threads (although this number of threads is not pre-populated) and it defaults to a value based on the number of logical cores. The maxThreads attribute specifies the maximum number of threads and defaults to unlimited. Liberty dynamically adjusts the thread pool size between coreThreads and maxThreads based on observed throughput.

coreThreads: This option specifies the core number of threads to associate with the executor of the thread pool... If this value is less than 0, a default value is used. This default value is calculated based on the number of hardware threads on the system. Tip: Start your tuning with coreThreads="5" for each hardware thread or logical processor. For example, for a two-core SMT-4 machine, which represents eight logical processors, use coreThreads="40" as a starting point.

maxThreads: Maximum number of threads that can be associated with the executor. If greater than 0, this value must be greater than or equal to the value of coreThreads. If the value of maxThreads is less than or equal to 0, the maximum number of threads is unbounded. Default value: -1

Without the maxThreads attribute specified, Liberty dynamically adjusts the thread pool size maximum based on observed throughput.

Example:

```
<executor maxThreads="100" />
```

Starting with Liberty 8.5.5.2, thread stealing has been disabled and the stealPolicy attribute is not applicable.

**Timed Operations**

When enabled, the timed operation feature tracks the duration of JDBC operations running in the application server. In cases where operations take more or less time to execute than expected, the timed operation feature logs a warning. Periodically, the timed operation feature will create a report, in the application server log, detailing which operations took longest to execute. If you run the server dump command, the timed operation feature will generate a report containing information about all operations it has tracked.

To enable timed operations, add the timedOperations-1.0 feature to the server.xml file.

The following example shows a sample logged message:


[3/14/13 14:01:25:960 CDT] 00000025 TimedOperation W  TRAS0080W: Operation websphere.datasource.execute: jdbc/exampleDS:insert into cities values ('myHomeCity', 106769, 'myHomeCountry') took 1.541 ms to complete, which was longer than the expected duration of 0.213 ms based on past observations.


<featureManager>
  ...
  <feature>timedOperations-1.0</feature>
</featureManager>

Timed operations was introduced before requestTiming and is largely superseded by requestTiming, although requestTiming only uses simple thresholds. Unless the more complex response time triggering is interesting, use requestTiming instead.

**Monitoring**

Liberty optionally exposes MXBeans for monitoring (disabled by default). To enable monitoring, add the monitor feature to server.xml: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/a/e/twlp_mon.html. Additionally, expose the JMX connector either to accesses from the same machine with the localConnector feature or remote accesses with the restConnector feature: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.multiplatform.doc/a/e/twlp_admin_jmx.html. For example, choose one of the following:

<featureManager>
  ...
  <feature>localConnector-1.0</feature>
  <feature>monitor-1.0</feature>
</featureManager>


The following are the minimum recommended MXBeans statistics to monitor:

- **JvmStats (e.g. WebSphere:type=JvmStats)**
  - Heap: Current Heap Size
  - UsedMemory: Current Heap Usage
  - ProcessCPU: Average percentage of CPU used over the previous interval by this JVM process
- **ServletStats (e.g. WebSphere:type=ServletStats,name=...)**
  - RequestCount: The cumulative number of processed requests.
  - ResponseTime (ns): Average response time of the servlet over the previous interval.
  - ThreadPoolStats (e.g. WebSphere:type=ThreadPoolStats,name=...)
  - ActiveThreads: The number of concurrent threads actively executing application-related work over the previous interval.

Page 361
- **PoolSize**: The current maximum size of the thread pool.
- **SessionStats** (e.g. WebSphere:type=SessionStats,name=...)
  - **LiveCount**: Total number of HTTP sessions cached in memory.
  - **ActiveCount**: The total number of concurrently active sessions. A session is active if Liberty is processing a request that uses that session.
- **ConnectionPool** (e.g. WebSphere:type=ConnectionPool,name=...)
  - **ManagedConnectionCount**: The number of ManagedConnection objects that are in use.
  - **ConnectionHandleCount**: The number of Connection objects that are in use.
  - **FreeConnectionCount**: The number of free connections in the pool.
  - **WaitTime**: The average waiting time in milliseconds until a connection is granted.

All MXBeans in a collective can be accessed by connecting to the collective controller using the JMX REST Connector and the RoutingContext Mbean.

Note that while JConsole does have some basic capabilities of writing statistics to a CSV, this is limited to a handful of JVM statistics from the main JConsole tabs and is not available for the MXBean data. If all MXBeans are enabled, IBM benchmarks show about a 4% overhead. This may be reduced by limiting the enabled MXBeans; for example:

```xml
<monitor filter="ServletStats,ConnectionPool,..."/>
```

To connect remotely with the restConnector, launch the client JConsole as follows:

```bash
jconsole -J-Djava.class.path=$JDK/lib/jconsole.jar:$JDK/lib/tools.jar:
$LIBERTY/clients/restConnector.jar -J-
-Djavax.net.ssl.trustStore=$LIBERTY/usr/servers/server1/resources/security/key.jks -J-
-Djavax.net.ssl.trustStorePassword=$KEYSTOREPASSWORD -J-
-Djavax.net.ssl.trustStoreType=jks
```

Then use a URL such as `service:jmx:rest://localhost:9443/IBMJMXConnectorREST` and enter the administrator credentials.

**JConsole**

1. Start JConsole: `${WLP}/java/${JAVA}/bin/jconsole`
2. Choose the JVM to connect to:
3. You may be asked to automatically switch to the secure port:

![Secure connection failed. Retry insecurely?](image)

The connection to 5409 could not be made using SSL.
Would you like to try without SSL?
(Username and password will be sent in plain text.)

4. Review the overview graphs:
5. Click the MBeans tab to review enabled data:

6. You may also export some of the data by right clicking and creating a CSV file:
7. You may also execute various MBeans, for example:
HTTP

maxKeepAliveRequests of httpOptions: This option specifies the maximum number of persistent requests that are allowed on a single HTTP connection if persistent connections are enabled. A value of -1 means unlimited. This option supports low latency or high throughput applications, and SSL connections for use in situations where building up a new connection can be costly. Here is an example of how you code this option in the server.xml file:

```
<httpOptions maxKeepAliveRequests="-1" />
```

To decrease response time of servlets, add the following attribute to the server.xml file:

```
<webContainerskipMetaInfResourcesProcessing="true"/>
```


Liberty (starting in 18.0.0.2) uses DirectByteBuffers for HTTP reading and writing just like traditional WAS; however, there is only a global pool rather than ThreadLocal pools, and the DBB sizes and bucket sizes may be configured with, for example:

```
<bytebuffer poolSizes="32,1024,8192,16384,24576,32768,49152,65536"
```
HTTP Sessions

By default, Liberty sets allowOverflow="true" for HTTP sessions, which means that maxInMemorySessionCount is not considered and HTTP sessions are unbounded which may cause OutOfMemoryErrors in the default configuration without session persistence into something like WebSphere eXtreme Scale. If allowOverflow is disabled, maxInMemorySessionCount should be sized taking into account the maximum heap size, the average HTTP session timeout, and the average HTTP session heap usage.

Java Database Connectivity (JDBC)

maxPoolSize of connectionManager: This option specifies the maximum number of physical connections for the connection pool. The default value is 50. The optimal setting here depends on the application characteristics. For an application in which every thread obtains a connection to the database, you might start with a 1:1 mapping to the coreThreads attribute. Here is an example of how you code this option in the server.xml file:

<connectionManager ... maxPoolSize="40" />

purgePolicy of connectionManager: This option specifies which connections to destroy when a stale connection is detected in a pool. The default value is the entire pool. It might be better to purge only the failing connection. Here is an example of how you code this option in the server.xml file:

<connectionManager ... purgePolicy="FailingConnectionOnly" />

numConnectionsPerThreadLocal of connectionManager: This option specifies the number of database connections to cache for each executor thread. This setting can provide a major improvement on large multi-core (8+) machines by reserving the specified number of database connections for each thread. Using thread local storage for connections can increase performance for applications on multi-threaded systems. When you set numConnectionsPerThreadLocal to 1 or more, these connections per thread are stored in thread local storage. When you use numConnectionsPerThreadLocal, consider two other values:

The number of application threads

The connection pool maximum connections

For best performance, if you have N application threads, set maximum pool connections to at least n times the value of numConnectionsPerThreadLocal attribute. For example, if you use 20 application threads, set the maximum pool connections to 20 or more. If you set the value of numConnectionPerThreadLocal attribute as 2 and there are 20 application threads, set the maximum pool connection to 40 or more. Here is an example of how you code this
option in the server.xml file:

```
<connectionManager ... numConnectionsPerThreadLocal="1" />
```

statementCacheSize of dataSource: This option specifies the maximum number of cached prepared statements per connection. To set this option, complete the following prerequisite:

Review the application code (or an SQL trace that you gather from the database or database driver) for all unique prepared statements.

Ensure that the cache size is larger than the number of statements.

Here is an example of how you code this option in the server.xml file:

```
<dataSource ... statementCacheSize="60" />
```

isolationLevel of dataSource: The data source isolation level specifies the degree of data integrity and concurrency, which in turn controls the level of database locking. Four different options are available as following in order of best performing (least integrity) to worst performing (best integrity).

- TRANSACTION_READ_UNCOMMITTED: Dirty reads, non-repeatable reads, and phantom reads can occur.
- TRANSACTION_READ_COMMITTED: Dirty reads are prevented; non-repeatable reads and phantom reads can occur.
- TRANSACTION_REPEATABLE_READ: Dirty reads and non-repeatable reads are prevented; phantom reads can occur.
- TRANSACTION_SERIALIZABLE: Dirty reads, non-repeatable reads, and phantom reads are prevented.

Here is an example of how you code this option in the server.xml file:

```
<dataSource ... isolationLevel="TRANSACTION_READ_COMMITTED">
```

e.doc/ae/twlp_tun.html

The DB2 JCC type 4 driver may create excessive Timer objects (if profiling shows hotspots in com.ibm.db2.jcc.am.Connection.scheduleQueryTimer) to manage per-statement query timeouts. The alternative connection-level timeout management has less overhead and may be configured with:

```
<properties.db2.jcc timerLevelForQueryTimeOut="2" />
```

See
Admin Center
The Admin Center is commonly put on port 9443, for example https://localhost:9443/adminCenter/

```xml
<featureManager>
  ...
  <feature>adminCenter-1.0</feature>
</featureManager>

<httpEndpoint host="*" httpPort="9080" httpsPort="9443"> ...
<quickStartSecurity userName="wsadmin" userPassword="wsadmin" />
```

Idle CPU
Reducing idle server CPU time: To reduce idle server CPU time, add the following attributes to the server.xml file:

```xml
<applicationMonitor dropinsEnabled="false" updateTrigger="disabled"/>
<config updateTrigger="disabled"/>
```

Authentication Cache

Because the creation of an authentication subject can increase processor usage, the Liberty profile provides an authentication cache to store a subject after the authentication of a user is successful. To fully take advantage of this service to increase performance, you must make sure that it is turned on and tuned according to your users and applications... By default, the authentication cache is enabled to help improve performance.

Consider changing the authentication cache timeout value. Increasing the timeout value enables subjects to remain in the authentication cache longer and reduces the number of reauthentications needed. However, increasing the timeout value increases the risk of user permissions becoming stale compared to a modified external repository, such as LDAP. Set your authentication cache timeout to reflect the estimated length of client sessions. You can specify the cache timeout by setting the value of the timeout attribute to whichever time you choose in the authCache element in the server.xml file. The default value is 600 seconds.

Finally, if you are experiencing authentication times longer than expected, or you are noticing more traffic to an external authentication repository than expected, the authentication cache might be full. When the authentication cache is full, subjects are evicted. There is not a one-to-one mapping of authenticated users to authentication cache entries. The number of entries in the cache per user depends on other security configurations. It is a best practice for the maximum size of the authentication cache to be larger than the number of distinct authenticated users that are accessing the server at one time. Setting the maximum size of the authentication cache this way helps prevent subjects from being evicted from the cache before timing out. You can change the maximum size of the authentication cache by setting the value of the maxSize attribute in the authCache element in the server.xml file. The default size is 25000.


LDAP

attributesCache

<size>: Specifies the number of entities that are stored in the cache. You can increase the size of the cache based on your business requirement, for example, increase the cache size if more number of entities are required in a business scenario.

<timeout>: Specifies how long the results can be cached before they are invalidated. If the back-end LDAP data is refreshed frequently to maintain an up-to-date cache, set a lesser
timeout duration value.

<sizeLimit>: Specifies the maximum number of LDAP attributes per entity that can be stored in the cache. If an entity is associated with many attributes, increase the <sizeLimit> value.

searchResultSizeLimit: Specifies the maximum number of search results that can be stored in the cache. Use the parameters in the <searchResultSizeLimit> element to tune the search results that are returned as part of the query.

You can adjust the following parameters in the <contextPool> element to control the cache:

contextPool

<initialSize>: Specifies the initial size of the context pool. The value must be set based on the load on the repository. If the initial number of requests to the LDAP server is expected to be high, increase the value of the initial size.

<maxSize>: Specifies the maximum context pool size. The value must be set based on the load on the repository. If you want to restrict the number of connections to the LDAP server, then set the value of the <maxSize> element to less than half of the maximum number of connections that the LDAP server can handle.

<timeout>: Specifies the duration after which the context pool times out. Specify a shorter timeout value so that fresh connections can be made to the LDAP server after the specified duration is timed out. For example, if the established connection is timed out after the configured interval, then set a shorter duration than the firewall timeout duration so that the connection is re-established.

<waitTime>: Specifies the waiting time before the context pool times out. If the value specified is high, then the time that is taken to establish a connection to the LDAP server is increased accordingly.

WAS Liberty on z/OS

"A Liberty Profile server instance may be started as a UNIX Systems Services process or as a started task. As a started task a Liberty Profile server instance appears as a single address space... The "Angel" process provides an anchor point for access to z/OS authorized services." (http://www-01.ibm.com/support/docview.wss?uid=tss1wp102110&aid=3)

Configuration Analysis

WebSphere Application Server Configuration Visualizer


WebSphere Application Server Configuration Comparison Tool

https://www-01.ibm.com/support/docview.wss?uid=swg22010928

IBM Visual Configuration Explorer (VCE)

The IBM Visual Configuration Explorer (VCE) tool is available for free in the IBM Support Assistant version 4.1 workbench. It is not currently available for ISA 5. The VCE tool only works on Traditional WAS configurations and does not currently support Liberty.

Gathering a VCE output file

Unix

2. Choose WebSphere Application Server
3. Click "Launch or Download IBM Support Assistant Data Collector for WebSphere Application
4. This will pop up a new window (make sure your pop-up blocker is disabled)
5. Select the radio button "this or another system using a downloadable utility"
6. Download ApplicationServer_2.0.9.tar.gz for Unix
7. Upload ApplicationServer_2.0.9.tar.gz to the deployment manager node(s).
8. Unpack ApplicationServer_2.0.9.tar.gz into a temporary directory
9. cd ${WebSphere}/AppServer/bin/
10. source ./setupCmdLine.sh # Note: this and the above command should be run separately. May not work with sudo.
11. Change directory to the temporary directory with ISADC
12. ./isadc.sh
13. Follow the instructions and select option "[2] General" followed by "[3] VCE Collector"
14. When asked, choose the deployment manager profile and node
15. This will take 1 CPU core for a few minutes, so it is best to do it on production during low load.

Windows
2. Choose WebSphere Application Server
3. Click "Launch or Download IBM Support Assistant Data Collector for WebSphere Application Server"
4. This will pop up a new window (make sure your pop-up blocker is disabled)
5. Select the radio button "this or another system using a downloadable utility"
6. Download ApplicationServer_2.0.11.zip for Windows
7. Upload ApplicationServer_2.0.11.zip to the deployment manager node(s).
8. Unpack ApplicationServer_2.0.11.zip into a temporary directory
9. cd %WebSphere%\AppServer\bin\
10. setupCmdLine.bat
11. Change directory to the temporary directory with ISADC
12. Run isadc.bat
13. Follow the instructions and select option "[2] General" followed by "[3] VCE Collector"
14. When asked, choose the deployment manager profile and node
15. This will take 1 CPU for a few minutes, so it is best to do it on production during low load.

Compare Configurations
Create a new Workspace > In Configurations > Right Click > Add Configuration
Load both .configuration files
You should see something like the following:
In the Object Compare Results view, VCE will show all differences (you may need to explicitly match differently named cells/nodes/servers in any "unresolved" nodes):

Log Analysis

IBM Support Assistant 5

ISA 5 has a very powerful log analysis engine built-in. Here is how to load and analyze log files:

1. In the top left, click Cases > Add
2. Enter a Summary and click the green checkbox
3. Click the "< Cases" button at the top right of the pop-up to hide it
4. Your case should now be selected in the cases dropdown box
5. In the Files tab, click the Add files button and select any log files such as SystemOut.log, a ZIP file of logs, etc.
6. If you uploaded a ZIP file, right click it and select Unpack
7. Click the "Scan this Case" button in the top right and click Submit
8. Once the scan completes, click the "Overview" and "Symptoms" tabs to review the log analysis.

Major Tools

This chapter will cover what we consider the most important performance analysis tools for the majority of situations. We cover other tools in other chapters and other tools may be the most important performance tool for a particular situation; however, for these tools we will generally cover them in more depth.
Eclipse

Eclipse is a free open source project that's used to run many of the major tools in this cookbook.

Installation

1. Install Eclipse such as Eclipse Neon: [http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a](http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a)

Setting the Java Virtual Machine that Eclipse Uses

Some tool usages require IBM Java to properly run the analysis (e.g. IBM MAT, IBM Java Health Center, etc.). You may specify the JVM that Eclipse uses in the eclipse.ini file. Above the -vmargs line, add the following two lines, replacing the path to IBM Java that you installed:

```
-vm
/opt/IBM/Java/ibm-java-x86_64-80/bin/
```

For Windows, IBM Java provides a pre-packaged Eclipse Neon with IBM Java already configured

2. If you are running Windows 32-bit, click on the "Windows on Intel" link:


3. If you are running Windows 64-bit, click on the "Windows on AMD64/EMT64T" link:


4. The download will require you to either register for a free IBM ID or use the "Proceed without an IBM id" button and enter your information.
5. When you get to the download page, the default option is to use the "Download Director" to download the file which is a Java applet that downloads more quickly by using multiple sockets; however, you may choose the simpler option of a direct link by click on the "Download using http" tab.

6. Extract the .zip file into a directory of your choice.

7. Go to the "eclipseDevelopmentPackage\eclipse" subdirectory and launch eclipse.exe.

**IBM Java on Linux**

1. Open a browser to https://developer.ibm.com/javasdk/downloads/#tab_sdk8

2. If you are running Linux 32-bit, under the "Linux on Intel" heading, click on the first “Simple unzip with license” link.

3. If you are running Linux 64-bit, under the "Linux on AMD64/EMT64T" heading, click on the first “Simple unzip with license” link.

4. Download the package to any directory, most commonly /opt/IBM/Java

5. From the terminal, add execute permissions and then run ibm-java-sdk-8.0-3.21-x86_64-archive.bin:

```
chmod +x ./ibm-java-sdk-8.0-3.21-x86_64-archive.bin
./ibm-java-sdk-8.0-3.21-x86_64-archive.bin
```

**Eclipse Maximum Heap Size**

The maximum heap size for Eclipse may be set in the "-vmargs" section of the eclipse.ini file.

**Offline Update Site Installation**

1. Download Eclipse update sites:

   1. Create a directory for the Eclipse download sites. For example C:\Program Files\Eclipse\updatesites (Windows) or /opt/eclipse/updatesites (Linux)

   2. Open a terminal (Windows cmd.exe or Linux command line) to the expanded Eclipse directory that you downloaded. For example:

      **Windows**:
      ```
cd "C:\Program Files\Eclipse\Neon\eclipse\"
      or
      cd "C:\Program Files\eclipseDevelopmentPackage\eclipse\"
      ```

      **Linux**:
      ```
cd /opt/eclipse/neon/eclipse/
      ```

   3. Check that you have Java on your path:

      ```
      > java -version
      java version "1.8.0"
      ```
Java(TM) SE Runtime Environment...

1. If this returns "java is not recognized as an internal or external command..." (Windows) or "bash: java: command not found" (Linux), then you must put Java on the PATH. For example:

   **Windows:**
   ```
   set PATH=%PATH%;"C:\Program Files\Eclipse\Neon\eclipseDevelopmentPackage\ibm_sdk80\bin"
   or
   set PATH=%PATH%;"C:\Program Files\Java\jdk1.8.0_66\bin"
   ```

   **Linux:**
   ```
   export PATH=${PATH}:/opt/IBM/Java/ibm-java-x86_64-80/bin/
   ```

4. Run the following two commands, replacing the target directory with the directory you created in step #1 above followed by wasdev. For example:

   **Windows:**
   ```
   ```
   and
   ```
   ```

   **Linux:**
   ```
   ```
   and
   ```
   ```

5. Run the following two commands, replacing the target directory with the directory you created in step #1 above followed by ibmtools. For example:
Windows:
eclips.exe -application org.eclipse.equinox.p2.metadata.repository.mirrorApplication
-destination "file:/C:/Program Files/Eclipse/updatesites/ibmtools/" -ignoreErrors

and

eclips.exe -application org.eclipse.equinox.p2.artifact.repository.mirrorApplication
-destination "file:/C:/Program Files/Eclipse/updatesites/ibmtools/" -ignoreErrors

Linux:
./eclipse -application org.eclipse.equinox.p2.metadata.repository.mirrorApplication
-destination "file:/opt/eclipse/updatesites/ibmtools/" -ignoreErrors

and

./eclipse -application org.eclipse.equinox.p2.artifact.repository.mirrorApplication
-destination "file:/opt/eclipse/updatesites/ibmtools/" -ignoreErrors

6. Run the following two commands, replacing the target directory with the directory you created in step #1 above followed by ibmtoolsiema. For example:

Windows:
eclips.exe -application org.eclipse.equinox.p2.metadata.repository.mirrorApplication
-destination "file:/C:/Program Files/Eclipse/updatesites/ibmtoolsiema/" -ignoreErrors

and

eclips.exe -application org.eclipse.equinox.p2.artifact.repository.mirrorApplication
-destination "file:/C:/Program Files/Eclipse/updatesites/ibmtoolsiema/" -ignoreErrors

Linux:
./eclipse -application org.eclipse.equinox.p2.metadata.repository.mirrorApplication
-destination "file:/opt/eclipse/updatesites/ibmtoolsiema/" -ignoreErrors

and

./eclipse -application org.eclipse.equinox.p2.artifact.repository.mirrorApplication
7. Run the following two commands, replacing the target directory with the directory you created in step #1 above followed by neon. This download is very large: about 4GB. For example:

**Windows:**

and


**Linux:**

and


2. Transfer the downloaded updatesites above to the target machine.

3. For each updatesite folder:

   1. In Eclipse, click Help → Install New Software…
   2. Click the "Add..." button.
   3. Enter "Local NAME" (for example, Local neon) for the name, click "Local..." and select the update site folder.
   4. After you click OK, Eclipse will automatically select the new update site and load available plugins from which you can select the plugins in the instructions in the main part of the lab.

**gnuplot**

This cookbook references scripts that use the open source gnuplot tool to generate graphs:

http://www.gnuplot.info/
graphcsv.gpi

The primary gnuplot script used is at

This is combined with the following script to generate the multiplot commands:

Some common things you may consider changing:

1. Uncomment the following line in graphcsv.gpi to produce a text-based graph to the console:
   #set terminal dumb
2. Uncomment the following lines in graphcsv.gpi to produce a PNG:
   #set terminal png
   #set output "output.png"
3. Remove "pause -1" from graphcsv.sh to disable the requirement to hit Ctrl+C after the graph is produced (this is particularly useful for #1 and #2 above)

Test Graphing

Test graphing with the following set of commands:

```
$ cat > data.csv
Time (UTC),CPU,Runqueue,Blocked,MemoryFree,PageIns,ContextSwitches,Wait,Steal
2014-10-15 16:12:11,20,0,0,12222172,0,2549,0,0
2014-10-15 16:12:12,27,1,0,12220732,0,3619,0,0
2014-10-15 16:12:13,30,0,0,12220212,0,2316,0,0
Ctrl+D
$ gnuplot -e "\n> set timefmt '%Y-%m-%d %H:%M:%S';
> set xdata time;
> set style data lines;
> set format y '%.0f';
> set datafile sep ',';
> set key autotitle columnhead;
> set multiplot layout 4,2 scale 1.0,0.8;
> plot 'data.csv' using 1:2;
> plot 'data.csv' using 1:3;
> plot 'data.csv' using 1:4;
> plot 'data.csv' using 1:5;
> plot 'data.csv' using 1:6;
> plot 'data.csv' using 1:7;
> plot 'data.csv' using 1:8;
> plot 'data.csv' using 1:9;
> unset multiplot;
> pause -1;"
Warning: empty y range [0:0], adjusting to [-1:1]
Warning: empty y range [0:0], adjusting to [-1:1]
Warning: empty y range [0:0], adjusting to [-1:1]
Warning: empty y range [0:0], adjusting to [-1:1]
```

Example output:
Garbage Collection and Memory Visualizer (GCMV)

The IBM Garbage Collection and Memory Visualizer (GCMV) tool is used primarily to analyze Java
memory usage using the output of verbose garbage collection. It is built by the IBM Java team and
fully supported, and parses both IBM Java and Oracle Java verbose garbage collection log files. For
IBM Java in particular, it has an advanced engine which provides automated analysis and
recommendations from the data.

Use GCMV to help you:

• Monitor and fine tune Java heap size and garbage collection performance
• Check for memory leaks
• Size the Java heap correctly
• Select the best garbage collection policy
- Compare different runs (for example, with different garbage collection policies)

**Installation**

This tool is available for free as a plugin for Eclipse:

2. Click Help > Eclipse Marketplace...
3. Search for "gcmv" and press Go
4. Click Install
5. After installation completes, restart Eclipse
6. Click Window > Perspective > Open Perspective > Other...
7. Select GCMV and click OK.
8. Click on File > Load File and and load the verbose gc file

1. GCMV parses and plots various log types including **Verbose GC logs**, javacore.txt, verbosegc fight recorder, -Xtgc output, and native memory logs (output from ps, svmmon and Perfmon).


**Features and Benefits**

GCMV uses a powerful statistical analysis engine which provides guidance on improvements in these areas:

- Memory Leak Detection
  - Detect Java heap exhaustion and memory leaks
  - Detect "native" (malloc) heap exhaustion and memory leaks
- Optimizing garbage collection performance
  - Determine garbage collection overhead
  - Detect long or frequent garbage collection cycles and causes
  - Recommend settings to avoid long or frequent garbage collection cycles
  - Recommend optimum garbage policy
- Fine tuning of Java heap size
  - Determine peak and average memory usage
  - Recommend Java heap settings

GCMV provides a flexible user interface, making it possible to carry out further analysis of the data and to "drill down" into the causes of trends or data points of interest.

The GCMV graphical interface provides the following capabilities:

- Raw log, tabulated data and graph views
- Ability to zoom and crop graphs
• Ability to select data points in line plots and view in raw data
• Customize the graph by adding/removing data and changing display units
• Compare output from multiple logs
• Save data to jpeg or .csv files for export to spreadsheets
• Templates allow configuration to be saved
• Support for compressed files and rolling logs

**Analysis**

Primarily, you will review the line plot to observe garbage collection behavior, and click on the Report tab to review the proportion of time spent in garbage collection.

Observe in the following example that towards the end of the graph, the “Used heap (after global collection)” – the brown line – which is the amount of live Java heap after a full garbage collection finishes, has a pattern where it doesn't decrease much and it's near the heap size (blue line). This also correlates with a persistent increase in the “Total pause time” - the dark line. These are the classic signs of heap exhaustion.

![Graph showing garbage collection behavior](image)

**Proportion of time in garbage collection:**

<table>
<thead>
<tr>
<th>Proportion of time spent in garbage collection pauses (%)</th>
<th>1.31</th>
</tr>
</thead>
</table>

**Cropping Data Analysis**

Zooming does not crop the data analysis. To do so, specify a minimum and maximum value for the X-axis.
Hover your mouse over the approximate start and end points of the section of concern and note the times of those points (in terms of your selected X Axis type):

![Graph showing heap usage over time]

Enter each of the values in the minimum and maximum input boxes and press Enter on your keyboard in each one to apply the values. The tool will show vertical lines with triangles showing the area of the graph that you've focused on.
Click on the “Report” tab at the bottom and observe the proportion of time spent in garbage collection for just this period (in this example, 87%).

**Customizing the Views**

**Adding and removing line plots**

Check or uncheck line plots from the "Data Selector" tab. For example, it is often useful to select VGC Pause > Total pause time, and VGC Heap > Used heap (after global collection).
X-axis

It is often useful to change the X-axis to date/time:

Zooming

You may zoom into any part of the graph, or reset the zoom in the Zoom view. Note that zooming does not affect the report (see the cropping section above for how to do that):
Comparing Different Runs

First, load the baseline verbosegc as normal. Next, right click anywhere in the plot area and click 'Compare File...':
Next, ensure that the X-axis uses a relative format such as hours, instead of date. Otherwise, you'll just end up essentially combining the two verbosegc files with a date gap in between and so you won't be able to visualize any differences.

Finally, zoom in to the part where they overlap (i.e. one might be longer than another, so cut the extra off). Important note: GCMV's zoom feature is only a different visualization of the line plot -- it does not affect GCMV's report tab. That means that if something went wrong outside your zoom which you don't care about, zooming in to avoid that section will not disregard that errant data in the report tab (for things such as proportion of time spent in GC, largest allocation, etc.). To do this, you'll also want to change the Minimum and Maximum X values in the Axes view to approximately match your zoom. It is easiest to first change the X-axis, at which point GCMV will gray out the disregarded data. Then, you can zoom around the non-disregarded data using your cursor.
For each series, there will be a solid line for the baseline verbosegc and a dashed line of the same color for the compared verbosegc. When you click on the report tab, GCMV will create a column for each verbosegc for easier comparison:
<table>
<thead>
<tr>
<th>Variant</th>
<th>native_stderr.log</th>
<th>native_stderr_2011_06_10_23_50_11.log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent collection count</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td>Forced collection count</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>GC Mode</td>
<td>gencon</td>
<td>gencon</td>
</tr>
<tr>
<td>Global collections - Mean garbage collection pause (ms)</td>
<td>518</td>
<td>593</td>
</tr>
<tr>
<td>Global collections - Mean interval between collections (minutes)</td>
<td>65.4</td>
<td>19.6</td>
</tr>
<tr>
<td>Global collections - Number of collections</td>
<td>15</td>
<td>26</td>
</tr>
<tr>
<td>Global collections - Total amount tenured (MB)</td>
<td>23522</td>
<td>43956</td>
</tr>
<tr>
<td>Largest memory request (bytes)</td>
<td>6041040</td>
<td>6860672</td>
</tr>
<tr>
<td>Number of collections triggered by allocation failure</td>
<td>9674</td>
<td>16539</td>
</tr>
<tr>
<td>Nursery collections - Mean garbage collection pause (ms)</td>
<td>123</td>
<td>155</td>
</tr>
<tr>
<td>Nursery collections - Mean interval between collections (ms)</td>
<td>8208</td>
<td>1841</td>
</tr>
<tr>
<td>Nursery collections - Number of collections</td>
<td>9668</td>
<td>16536</td>
</tr>
<tr>
<td>Nursery collections - Total amount tenured (MB)</td>
<td>72813</td>
<td>131466</td>
</tr>
<tr>
<td>Nursery collections - Total amount tenured (MB)</td>
<td>8531</td>
<td>16073</td>
</tr>
<tr>
<td>Proportion of time spent in garbage collection pauses (%)</td>
<td>1.48</td>
<td>7.82</td>
</tr>
<tr>
<td>Proportion of time spent unpaused (%)</td>
<td>98.52</td>
<td>92.18</td>
</tr>
<tr>
<td>Rate of garbage collection (MB/minutes)</td>
<td>313</td>
<td>1288</td>
</tr>
</tbody>
</table>

In this case, we can see that, for example, the proportion of time spent in GC went from 7.82% to
1.48% (the native_stderr.log was the newer one). Many of the other statistics got better. In this case, we can say that the tuning we did (increasing the nursery size) was very beneficial, all other things being equal.

Now one very important consideration is "all other things being equal." You have to be very careful comparing verbosegc. If, for example, a different amount or rate of work came into these independent runs (for example, a different test was run, or one day was a workday and another a weekend with less work, etc.), then it would be much more difficult to conclude anything. One obvious sign of this is that you're tuning something like the nursery, and the overall Java heap usage is magnitudes different. The point is: carefully control your experiment to hold all other variables constant (and verify using data such as request count, response times, etc.).

IBM Thread and Monitor Dump Analyzer (TMDA)

Overview
The IBM Thread and Monitor Dump Analyzer (TMDA) tool analyzes IBM and Oracle thread dumps, extracting out thread stacks and monitors and displaying them in a GUI.

Thread dumps are primarily used in performance analysis as a low frequency, sampling profiler. They are a lightweight and generally non-intrusive way to get a picture of what the JVM is doing. For more details about profilers, see the Java Profilers chapter. The IBM Java and Oracle Java troubleshooting sections list all of the ways to generate thread dumps.

Use TMDA to help you:

- Get a picture of what the JVM is doing
- See how threads are moving (or not moving) over time using the thread comparison view
- Check for deadlocks or lock contention

Features and Benefits
TMDA displays thread dumps in an easy-to-navigate GUI view:

- Color code threads based on run state
- Summarize threads by the top stack frames
- Show a tree view of monitors
- Analyze native memory information in IBM Javacores

Installation
TMDA is available for free as a standalone tool at https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=2245aa39-fa5c-4475-b891-14c205f7333c

1. Click the Download Now link
2. Download jca*.jar to any local directory.
3. Ensure that a Java Runtime Environment is on your PATH and double click the JAR file to launch it.
4. If it fails to launch, run the command "java -version" from a command prompt or terminal. If
this gives an error, then Java is not on your path. You may set it as follows:

Windows:
set PATH=%PATH%;"C:\Program Files\Java\jdk1.8.0_66\bin"

or
set PATH=%PATH%;"C:\Program Files\Eclipse\Neon\eclipseDevelopmentPackage\ibm_sdk80\bin"

Linux:
export PATH=${PATH}:/opt/IBM/Java/ibm-java-x86_64-80-80/bin/

5. Then execute TMDA with "java -jar jca457.jar"

Usage

1. Click the Open Thread Dumps button to load the thread dump files:

![Open Thread Dumps](image)

2. Select one or more thread dumps from the list and click on **Analysis**.

![Thread Dump List](image)

3. You can do the following analysis:
1. Native memory analysis
2. Thread detail analysis
3. Details of the thread monitor

**Compare threads from different files**

Select one or more thread dumps in the thread dump list and click the Compare Threads button (also works for a single thread dump):

There will be a lot of threads that will be irrelevant in 90% of cases. Focus on the threads where your application work occurs, such as the WebContainer thread pool. In this example, all of the threads are waiting for work (either parked in the WAS BoundedBuffer or in IBM AIO code waiting for the next event). Remember that only the full stack is meaningful. In some cases, a parked thread, or a thread waiting in Object.wait may be a problem, so it's best to look methodically through the stacks.

Key things to look for are:

- Are there any patterns in the stacks? For example, do you see a particular application stack frame always at the top? Or do you see a particular application stack frame somewhere in the middle that suggests that one particular function is slow?
- Are some or most of the threads waiting on a backend service such as a database or web service? If so, can you figure out from the stacks if these are coming from a particular application function?

<table>
<thead>
<tr>
<th>Thread Name</th>
<th>State</th>
<th>WebContainer : 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebContainer : 5</td>
<td>Runnable</td>
<td>at java/net/SocketInputStream.socketRead0(Native Method)</td>
</tr>
<tr>
<td>WebContainer : 4</td>
<td>Runnable</td>
<td>at java/net/SocketInputStream.read(SocketInputStream.java:164(Compiled Code))</td>
</tr>
<tr>
<td>WebContainer : 3</td>
<td>Runnable</td>
<td>at java/net/SocketInputStream.read(SocketInputStream.java:134(Compiled Code))</td>
</tr>
<tr>
<td>WebContainer : 2</td>
<td>Runnable</td>
<td>at com/ibm/io/asy...</td>
</tr>
<tr>
<td>WebContainer : 1</td>
<td>Runnable</td>
<td>at sun/misc/Unsafe...</td>
</tr>
<tr>
<td>WebContainer : 0</td>
<td>Runnable</td>
<td>at java/net/Socket...</td>
</tr>
</tbody>
</table>
Monitor analysis is also important to find Java lock bottlenecks. Click the Monitor Detail or Compare Monitors buttons to explore the hierarchy of blocked threads. Remember that some blocked threads are normal, such as threads in a thread pool waiting for the next piece of work.

**IBM Memory Analyzer Tool (MAT)**

**Overview**


**Standalone Installation**

Download and install the Eclipse Memory Analyzer Tool and then add the IBM DTFJ and IBM IEMA extensions on top of it:

2. Extract the compressed file and launch the MemoryAnalyzer executable.
3. Click Help → Install New Software…
5. Check "IBM Monitoring and Diagnostic Tools", click Next, click Next again, click "I accept..." if the license terms are okay, and click Finish.
6. Note that there may be a popup asking you to trust a certificate from IBM Canada. Make sure to check the box next to the certificate name before continuing.
7. After the plugins are installed, click "Yes" to restart Eclipse.
8. After Eclipse has been restarted, click Help → Install New Software…
10. Check "IBM Runtime Monitoring and Diagnostic Tools", install the plugins and restart Eclipse.
11. Note: Recent versions of DTFJ require Java >= 8. If that's not already on your PATH, install such a JDK and then point Eclipse to it by editing the .ini file and adding:

    -vm
    <path_to_java8_sdk>/bin/java

**Usage**

1. Click File > Open Heapdump and locate the IBM PHD Heapdump (.phd), Operating system core dump (.dmp), or Oracle HPROF heapdump (.hprof).
2. Note that the parser is determined by the file extension, so it is important to have the right extension. For example, core dumps might be missing the .dmp extension, so just rename the file with a .dmp extension and MAT will see it.
**First Dialog**

After a dump is loaded, a dialog will appear suggesting to run various reports such as the leak suspects report. If the dump is from an OutOfMemoryError, then go ahead and run that report if you'd like; otherwise, simply click Cancel to browse the dump normally:

![Image of the Getting Started Wizard dialog]

**Common Tasks**

The Overview tab shows:

A) How much heap is used at the time of the dump (MAT performs a full garbage collection when loading the dump, so this does not include any garbage)

B) The largest dominator objects
C) If the IBM Extensions for Memory Analyzer are installed, a link to the WAS Overview report that will provide a WAS-centric view of the dump

D) A histogram shows the heap usage by class

E) The dominator tree shows the heap usage by dominator objects in an expandable tree view

F) Top consumers shows heap usage by package.

G) Open Query Browser provides many advanced ways to look at the data, and also most of the IBM Extensions for Memory Analyzer plugins

H) The leak suspects report will search for likely causes of a leak in the dump.

### Object Query Language (OQL)


Select java.io.File objects that contain a string in their path:

```sql
select * from java.io.File f where toString(f.path).contains("IBM")
```
Select all threads that contain something in their name:

```sql
SELECT OBJECTS x FROM INSTANCEOF java.lang.Thread x WHERE x.toString().contains("WebContainer")
```

Select instances of some class which have a retained size > 24 bytes:

```sql
select * from instanceof com.ibm.MyClass s where s.@retainedHeapSize > 24
```

Select non-viewed, non-phantomed DirectByteBuffers:

```sql
SELECT k, k.capacity FROM java.nio.DirectByteBuffer k WHERE ((viewedBuffer=null)and(inbounds(k).length>1))
```

Select dominators of all instances of some class:

```sql
SELECT DISTINCT OBJECTS dominatorof(x) FROM java.lang.String x
```

Select dominator names of Strings:

```sql
SELECT classof(dominatorof(s)).@name, s FROM java.lang.String s WHERE dominatorof(s) != NULL
```

Select Strings with dominators of a particular type:

```sql
SELECT * FROM java.lang.String s WHERE dominatorof(s) != NULL and classof(dominatorof(s)).@name = "com.ibm.Test"
```

Select all class instances of a particular type:

```sql
SELECT OBJECTS c FROM INSTANCEOF java.lang.Class c WHERE c.@displayName.contains("class org.apache.commons.logging.impl.Jdk14Logger ")
```

Select a field from static class instances:

```sql
SELECT c.controller FROM INSTANCEOF java.lang.Class c WHERE c.@displayName.contains("class com.ibm.ws.management.util.PlatformHelperImpl ")
```

**Heapdump Theory**

Retained Heap: It is guaranteed that all objects below an entry are retained or kept alive by the parent. If you assume that object is removed, then the rest have been GCed.

The retained set includes the objects referenced by the fields on the given objects and all objects which are lifetime-dependent on them, i.e. which would be garbage collected if the references at the given fields at the given objects would be nulled.

The dominator tree is a transformation of the graph which creates a spanning tree (all objects in the graph are also in the dominator tree), removes cycles, and models the keep-alive dependencies. Object domination is equivalent to object retention, i.e. the set of objects dominated by some object are the same as the retained set of that object ([http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.mat.ui.help%2Fconcepts%2Fshallowretainedheap.html](http://help.eclipse.org/luna/index.jsp?topic= %2Forg.eclipse.mat.ui.help%2Fconcepts%2Fshallowretainedheap.html))

A garbage collection root is an object which has a reference to it from outside the heap (for example,
Retained Sets

"Retained set of X is the set of objects which would be removed by GC when X is garbage collected." (http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.mat.ui.help%2Fconcepts%2Fshallowretainedheap.html)

When most people talk about the "size" of a set of objects X, they are really talking about the retained set of the set of objects X, i.e. if nothing referenced X, then those objects could be garbage collected and the number of bytes representing the retained set of X would be freed.

Therefore, if you want to know how much memory a set of objects retain, click Open Query Browser > Java Basics > Show as Histogram, specify the objects, and then click "Calculate retained size" and select either of the two options. For example, one common cause of excessive heap usage is by org.apache.jasper.runtime.BodyContentImpl objects due to the default behavior of com.ibm.ws.jsp.limitBuffer=false. If we want to see how much these buffers are retaining, we can show a histogram for BodyContentImpl and calculate a precise retained size, in this example 291MB:
Show As Histogram

Enter a class name pattern (java.util.*)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>objects</td>
<td>org.apache.jasper.runtime.BodyContentImpl</td>
</tr>
</tbody>
</table>

- include class instance (if defined by a pattern)
- include subclasses (if object is a class)
- include loaded objects (if specified object is a classloader)
- as retained set
  
  simple mode...

-groupby Group by class
Another way to do this is to simply open the class Histogram and then filter in the Class Name column and calculate the retained size:

One useful technique when first analyzing a heapdump is to open the class histogram, calculate minimum retained sizes (you probably don't want to do precise as there may be many classes), and then sort by the "Retained Heap" column. It's important to note that each retained heap value is exclusive of the other values, so don't add this column up. For example, we may see that char[] retain hundreds of MB and BodyContentImpl objects retain hundreds of MB, but in this example, the BodyContentImpl objects retain the char[] objects.

It's nice to know how "big" a set of objects is but it's even better to get a class histogram of what is in that retained set. To do that, either right click on a set of objects and select "Show Retained Set," or use Open Query Browser > Show Retained Set and specify the objects. One tip is that you can use wildcards, so if you want to know how much memory is retained by some set of classes (e.g. everything made by one vendor), simply do com.example.* and review the sum of shallow heaps (in this example, we can say org.apache classes retain 321MB).
It's important to understand the limitations of retained sets. Complex object graphs often complicate retained sets. For example, WAS classes such as `com.ibm.ws.webcontainer.httpsession.MemorySessionContext` hold all HTTP sessions, so you may think that you can get the size of all HTTP sessions by simply looking at the retained set of this class. However, let's say WebContainer threads are currently working on some set of HTTP sessions at the time of the heapdump. In that case, those sessions are not part of the retained set of `MemorySessionContext` because there are references to those objects from outside `MemorySessionContext`. For specific situations, MAT has a Customized Retained Set query where you can explicitly say which objects to exclude from the set of incoming references (in this example, you
would specify MemorySessionContext and specify the set of application objects that reference these sessions as the exclude list). An alternative way to answer the question of how big all the session are is to calculate the retained set of all of the actual session objects instead of the map that contains them.

**Class Histogram**

Sometimes you'll see a dump where there are no obvious causes of high memory usage in the dominator tree nor the top consumers report. For example, here is a dump retaining 4.2GB of Java heap without any large dominators:

<table>
<thead>
<tr>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: 4.2 GB Classes: 50.8k Objects: 26m Class Loader: 2.8k</td>
</tr>
</tbody>
</table>

![Class Histogram](image)

The top consumers report is equally uninteresting:

**Biggest Top-Level Dominator Packages**

<table>
<thead>
<tr>
<th>Package</th>
<th>Retained Heap</th>
<th>Retained Heap, %</th>
<th># Top Dominators</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;all&gt;</td>
<td>4.15 GB</td>
<td>100.00%</td>
<td>2,556,710</td>
</tr>
</tbody>
</table>

The leak suspects report is slightly more interesting. The suspect is a set of 730 instances of HashMap retaining 2.26GB; however, each individual HashMap is no more than 57MB:
Where do we go from here? We know it's something in HashMaps but there isn't a particular HashMap to look at. Let's go to the class histogram which shows heap usage grouped by class:

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Objects</th>
<th>Shallow Heap</th>
<th>Retained Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Regex&gt;</td>
<td>&lt;Numeric&gt;</td>
<td>&lt;Numeric&gt;</td>
<td>&lt;Numeric&gt;</td>
</tr>
<tr>
<td>char[]</td>
<td>3,410,696</td>
<td>2.99 GB</td>
<td>&gt;= 2.99 GB</td>
</tr>
<tr>
<td>java.util.HashMap$Entry[]</td>
<td>301,598</td>
<td>33.02 MB</td>
<td>&gt;= 2.60 GB</td>
</tr>
<tr>
<td>java.util.HashMap</td>
<td>231,397</td>
<td>10.59 MB</td>
<td>&gt;= 2.49 GB</td>
</tr>
</tbody>
</table>

Click the little calculator and select "Calculate Minimum Retained Size (quick approx)" to see approximately how much each class and its instances retain.

Next, right click on HashMap and select "Show Objects by class > incoming references." This will show all of the objects that have references to these HashMaps and group the objects by class:

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Objects</th>
<th>Shallow Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>char[]</td>
<td>3,410,696</td>
<td>2.99 GB</td>
</tr>
<tr>
<td>java.util.HashMap$Entry[]</td>
<td>301,598</td>
<td>33.02 MB</td>
</tr>
<tr>
<td>java.util.HashMap</td>
<td>231,397</td>
<td>10.59 MB</td>
</tr>
</tbody>
</table>

As we expand the top level element, again we'll want to calculate minimum retained size and look for the class and its instances that retains the most. In this case, it is a set of 4,933 instances of PageContextImpl retaining about 2GB of heap.
This is far enough, but just one last step will be interesting which is to right click on PageContextImpl and choose Show Retained Set:

This shows a histogram by class of the set of objects retained by the selection. We can see that most of the memory held by the PageContextImpl and HashMap objects is character arrays. This lines up with the histogram we saw for the whole heap above, and we could have just as quickly gotten to the root cause by simply starting at the histogram and showing incoming references by class on the top element.

Objects Held by Thread Stack Frames

Load an IBM Java system dump or a recent Oracle Java HPROF dump and open Thread Overview and Stacks:
Expand the relevant stack frames and review the stack frame locals:

```
Object / Stack Frame   | Retained Heap | Name
-----------------------|---------------|---------------------
com.ibm.ws.util.ThreadPool$Worker @ 0x4202f0 | 5,474,256     | WebContainer : 2
at com.ibm.ws.rsadapter.jdbc.WSjdbcObject.closeJV (WSjdbcObject) | 80 |
  <local> com.ibm.ws.rsadapter.jdbc.WSjdbcResultSet @ 0xf39f70 | 40 |
  <local> java.lang.String @ 0x8bebe48 select * from test.table1 | 24 |
  <local> com.ibm.DatabaseTest @ 0x1716f8 | 96 |
  <local> com.mysql.jdbc.ResultSetMetaData @ 0x107d2948 | 24 |
  <local> java.lang.OutOfMemoryError @ 0x4258d30 Thread | 288 |
  <local> java.lang.String @ 0x107d2970 test | 40 |
  <local> com.ibm.ws.rsadapter.jdbc.WSjdbcResultSet @ 0xf39f70 | 80 |
  <local> com.ibm.ws.rsadapter.jdbc.WSjdbcStatement @ 0x175e2 | 112 |
  <local> java.util.ArrayList @ 0x175b28 | 5,378,120 |
  <local> com.ibm.ws.rsadapter.jdbc.WSjdbcConnection @ 0x175b3 | 232 |
  <local> com.ibm.jsp.jdbc.jdbcjdbcService(Ljava/servlet/http/HttpServletRe | 11 |
  at com.ibm.jsp.jdbc.jdbcjdbcService(Ljava/servlet/http/HttpServletRe | 11 |
  at com.ibm.ws.jsp.runtime.HttpJspBase.service(Ljava/servlet/http/HttpS
```

Comparing Heap Dumps

Acquire two or more heap dumps from the same run of the same JVM process, load both heap dumps
in MAT, open the Histogram in the latest heap dump and then use the Compare to another Heap Dump button:

![Heap Dump Comparison](image)

This will show a comparison of the class histograms between the two dumps, sorted by shallow size. In the example below, the latest dump has 20MB more of byte arrays, although there are 19,145 fewer of them (this means that the average size of a byte array has increased). As with class histograms in general, you often want to skip past primitives, Strings, and collections, in this case taking us to 21,998 more instances of RemovedEntry, taking up 703,995 more bytes of shallow heap. At this point, there is no science to discovering the leak (unless it's obvious), but one approach would be to see if the "uncommon" classes are holding the "common" classes; i.e. do the RemovedReaper and TTLHeapEntry objects retain HashMap entries? We can see just by the object counts that it is likely, and therefore, those uncommon objects are a leak suspect.

Note that object addresses and identifiers may change between dumps:

Object IDs which are provided in the heap dump formats supported by MAT are just the addresses at which the objects are located. As objects are often moved and reordered by the JVM during a GC these addressed change. Therefore they cannot be used to compare the objects. This basically means that if one compares two different heap dumps (although from the same process) it is not possible to point to the concrete objects different between the two heap dumps. However, one can still perform comparison on the aggregated results (e.g. the class histogram) and analyze how the amount of object and the memory they take has changed.

MAT also has extended differencing capabilities beyond the class histogram with the compare basket:

**SoftReferences**

Even if you observe increasing heap utilization after global collection over time:
It is possible that this is caused by SoftReferences being allocated faster than they're being garbage collected. If this is the case, the JVM will clean up garbage SoftReferences if necessary:

All soft references to softly-reachable objects are guaranteed to have been cleared before the virtual machine throws an OutOfMemoryError. Otherwise no constraints are placed upon the time at which a soft reference will be cleared or the order in which a set of such references to different objects will be cleared. Virtual machine implementations are, however, encouraged to bias against clearing recently-created or recently-used soft references. ([http://docs.oracle.com/javase/7/docs/api/java/lang/ref/SoftReference.html](http://docs.oracle.com/javase/7/docs/api/java/lang/ref/SoftReference.html))

The rate at which soft references are cleared is controlled with -XsoftrefthresholdX (IBM Java) and -XX:SoftRefLRUPolicyMSPerMB=X (Oracle Java).

In MAT, you can see how much memory is only softly retained with Java Basics > References > Soft references statistics and review the Total line of the Shallow Heap column in the Only Softly Retained tab:
Index Files

Most of the index files are divided into compressed pages of bytes held by soft references, so when memory is short they can be discarded and then reloaded, so you would have to ensure the soft references weren't cleared if you skipped writing the files. The index writers create the pages and write them to disk, but then pass the pages and the file across the reader, so provided the pages are present the file might not be needed.

The parser builds some index files, then the garbage cleaner removes unreachable objects and rewrites
the indexes with the new identifiers and also builds some new index files including the inbound index. The inbound index does have an intermediate stage which is written to disk - the .log files, which are not held in memory. The rewriting also writes some of the index files in a different format e.g. the outbound index is written in order so that it just extends to the start of the outbound references for the next index.

The dominator tree stage releases all the index files as it needs a lot of space (at least 7 int arrays the size of the number of objects in the dump). You would need to make sure you had enough memory to hold everything.

**Unreachable Objects**

MAT performs a full garbage collection when it first loads a heapdump. On the Overview tab, if there was any garbage, there will be a link to the Unreachable Objects Histogram, which will provide a histogram of the garbage collected:

http://wiki.eclipse.org/index.php/MemoryAnalyzer/FAQ#How_to_analyse_unreachable_objects

**Source Code**

The MAT source code is here: https://git.eclipse.org/c/mat/org.eclipse.mat.git

**IBM Extensions for Memory Analyzer (IEMA)**

The IBM Extensions for Memory Analyzer are a set of product specific extensions for MAT (http://www.ibm.com/developerworks/java/jdk/tools/iema/) and are available for free as optional extensions in the IBM Support Assistant.

**Installation**

The tool is also available as a plugin in the Eclipse IDE:

1. Install Eclipse such as Eclipse Neon: [http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a](http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a)
2. Open Eclipse, click Help → Install New Software…
3. Click the "Work with:" selection box and select "--All Available Sites--". It will take a few minutes to populate available plugins (showing "Pending...").
4. Under Performance, Profiling and Tracing Tools, check "Memory Analyzer" and "Memory Analyzer (Charts) [optional]" and click Next. Click Next again, click "I accept..." if the license terms are okay, and click Finish.
5. After the plugins are installed, click "Yes" to restart Eclipse.
6. After Eclipse has been restarted, click Help → Install New Software…
7. Click the "Add..." button. Enter "IBM Tools" for the name and the following Location and click OK: http://public.dhe.ibm.com/ibmdl/export/pub/software/websphere/runtimes/tools/
8. Check "IBM Monitoring and Diagnostic Tools", click Next, click Next again, click "I accept..." if the license terms are okay, and click Finish.
9. After the plugins are installed, click "Yes" to restart Eclipse.
10. After Eclipse has been restarted, click Help → Install New Software…
12. Check the additional "IBM Monitoring and Diagnostic Tools", install the plugins and restart Eclipse.

IBM Java Health Center

IBM Monitoring and Diagnostics for Java - Health Center (http://www.ibm.com/developerworks/java/jdk/tools/healthcenter/) is free and shipped with IBM Java itself (and therefore with WebSphere on platforms that run IBM Java). Among other things, Health Center includes a statistical CPU profiler that samples Java stacks that are using CPU (to sample all stacks, see "Health Center Thread Stacks" below) at a very high rate to determine what Java methods are using CPU. Starting with Java 5 SR10 (WAS 6.1.0.27), Java 6 SR 5 (WAS 7.0.0.5), Java 626 (WAS 8+), and Java 7, Health Center generally has an overhead of less than 1% and is suitable for production use. In recent versions, it may also be enabled dynamically without restarting the JVM.

Health Center Recipe

1. Open Garbage Collection View > Right Click > Change units > X axis > Date
2. Observe the garbage collection to see roughly when there was significant activity and mouse over the start and end times. Take these times and crop the data in the menu > Data > Crop Data
3. Observe the proportion of time spent in garbage collection % in the GC Summary view
4. Switch to the CPU view and observe the average and maximum CPU %
5. Switch to the Locking view and sort by Slow. If the number of gets or average hold times are high for the top hitters, review the lock Name
6. Switch to the Profiling view
7. Sort by self % (default sort) and observe the top 10 methods or so. If a single method self % is greater than 5-10%, this is concerning. Click on each one and observe the Invocation paths.
8. Sort by tree %. Usually the first one will be something like Thread.run or Worker.run. Select this and change to the Called methods view. Expand the largest tree items until there is a large "drop;" for example, if methods are 100, 99, 100, 100, etc., and then suddenly there is a drop to one method with 60% and one with 40%, this is usually an indication of a major divergence in general application activity. Continue as needed until something interesting comes up (this is an
Client Installation

1. Install Eclipse such as Eclipse Neon: [http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a](http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon1a)

   2. Open Eclipse, click Help → Install New Software…
   4. Check "IBM Monitoring and Diagnostic Tools", click Next, click Next again, click "I accept..." if the license terms are okay, and click Finish.
   5. After the plugins are installed, click "Yes" to restart Eclipse.

Gathering Data

There are two ways to gather HealthCenter data: 1) socket mode, and 2) headless mode. Socket mode requires a direct TCP connection from a HealthCenter client to a JVM with the HealthCenter agent enabled (either by restarting with -Xhealthcenter or dynamically enabling it with the same option). Headless mode writes data to local .hcd files (subsequently opened with the HealthCenter client) with the HealthCenter agent enabled in headless mode (either by restarting with -Xhealthcenter:level=headless or dynamically enabling it with the same option). In general, headless mode is preferred to avoid connectivity/firewall issues and doesn't require direct human involvement, and it is the mode covered in the following TechNote. However, when using the socket mode, the HealthCenter client has an option to export captured data to an .hcd file.

The following is an excerpt for the latest versions of Java from a TechNote on enabling HealthCenter. For other versions, see the full TechNote at [http://www-01.ibm.com/support/docview.wss?uid=swg21657760](http://www-01.ibm.com/support/docview.wss?uid=swg21657760)

Java 6 >= SR13 (WAS >= 7.0.0.29), Java 626 >= SR4 (WAS >= 8.0.0.6), Java 7 >= SR4 (WAS >= 8.5.0.2)

1. The Health Center agent can be enabled by restarting the JVM with
-Xhealthcenter:level=headless (1a) or Health Center may be enabled dynamically without restarting (1b). Choose one of these modes, preferably 1(b) since it doesn't involve a restart (if you don't need to profile startup):

1a. Add -Xhealthcenter:level=headless to the generic JVM arguments (http://www-01.ibm.com/support/docview.wss?uid=swg21417365). Save, synchronize and restart the JVM.

1b. Run ${WebSphere}/java/bin/java -jar ${WebSphere}/java/jre/lib/ext/healthcenter.jar ID=${PID} level=headless

Note: For both 1a and 1b, you may add the following arguments to limit and roll the total file usage of Health Center data:
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.max.size=BYTES
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=N (N=0 for unlimited)

Example for up to 10 historical files of 1GB each:
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.max.size=1073741824
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=10

Commonly used: -Xhealthcenter:level=headless
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.max.size=1073741824
-Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=10

2. Reproduce the problem. Health Center will write temporary files to the current working directory (or tmp_* subdirectory) of the JVM (normally, the profile directory where javacores go)

3. Gracefully stop the JVM. Upon successful termination, the Health Center agent will zip any pending "Source" files into the last .hcd file. If you cannot gracefully stop the JVM or require the HCD file immediately, you may copy off the "Source" files and manually zip them and rename the zip file with an .hcd extension (the jar command can be used to create a zip file); however, this is an unsynchronized operation and is not guaranteed to work.

4. Upload the resulting *.hcd files and WAS logs to IBM support.

When using the socket mode, by default Health Center uses CORBA as the communication protocol. Another option is to use JRMP with -Dcom.ibm.java.diagnostics.healthcenter.agent.transport=jrmp. Both methods are similar in performance although JRMP has a built-in reaper thread that runs approximately once an hour which calls System.gc.

For all healthcenter configuration properties, see https://www.ibm.com/support/knowledgecenter/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/configproperties.html

**Disabling Health Center**

After data collection has started, the only way to turn off the data collection is if you are connected through socket mode. In the HealthCenter client, open the "Configure Collection of Data" screen and uncheck all boxes. This will send a command to the agent to turn off most collection. In headless mode, the only way to emulate this is to pass parameters that specify that the collection will only occur for a
specified period of time.

**Overhead**

In general, the overhead of Health Center is between 0.4 to 3%, depending on the mode. In the headless mode, the overhead is about 1%; however, if files roll over, this involves zipping files together and that has a momentary impact, which averaged out can increase overhead up to 3%. The socket mode has the lowest overhead of about 0.4%.

**Health Center overhead**

![Application Throughput](image)

Measured using WebSphere App Server and the DayTrader benchmark with 50 clients

Running WAS 8.5.5, IBM Java 7 SR5, AIX 7.1, POWER7

Throughput determined by number of completed transactions on 4 saturated CPUs

**Analyzing Data**

1. Start Eclipse
2. Click Window > Perspective > Open Perspective > Other...
3. Select Health Center Status Summary and click OK.
4. Click File > Load Data and select the .hcd file
**WARNING**: HealthCenter may drop data if its Java heap is nearly full, so use a large heap when analyzing.

**Profiling View**

Click the Profiling link. By default, the list is sorted by self percentage which will show very hot methods.

- **CPU**
- **Classes**
- **Environment**
- **Garbage Collection**
- **I/O**
- **Locking**
- **Method Trace**
- **Native Memory**
- **Profiling**
- **Threads**

**WebSphere Real Time**

The Self (%) column reports the percent of samples where a method was at the top of the stack. The Tree (%) column reports the percent of samples where a method was somewhere else in the stack. In this example, WAS NCSA access logging was sampled more than 2% of the time, and the next largest single sample is java.lang.String.regionMatches. If we click on this row, we can expand the “Invocation Paths” to see which methods call this method. In this example, this 1% of samples was mostly called by JSF HTML rendering from the application.
If you sort by Tree %, skip the framework methods from Java and WAS, and find the first application method. In this example, about 47% of total samples was consumed by `com.ibm.websphere.samples.daytrader.web.TradeAppServlet.performTask` and all of the methods it called. The “Called Methods” view may be further reviewed to investigate the details of this usage.

**Health Center Thread Stacks**

The profiler in Health Center is only aware of threads that use CPU, so if a thread is waiting on a database, for example, it will not show up in Health Center. However, starting with the Health Center agent version 2.2, it periodically captures every thread stack. The Health Center client has minimal capabilities to display this information; however, you can use the Health Center API ([https://www.ibm.com/support/knowledgecenter/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/api.html](https://www.ibm.com/support/knowledgecenter/SS3KLZ/com.ibm.java.diagnostics.healthcenter.doc/topics/api.html)) to read an HCD file and print these stacks:

```java
import java.io.File;
import java.io.PrintWriter;
import java.lang.Thread.State;
```
import java.lang.management.LockInfo;
import java.lang.management.MonitorInfo;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import com.ibm.java.diagnostics.healthcenter.api.HealthCenter;
import com.ibm.java.diagnostics.healthcenter.api.factory.HealthCenterFactory;
import com.ibm.java.diagnostics.healthcenter.api.threads.ThreadData;
import com.ibm.java.diagnostics.healthcenter.api.threads.ThreadsData;

public class HealthCenterThreadDumpAnalyzer {
    private static final SimpleDateFormat threadDumpDate = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

    public static void main(String[] args) throws Throwable {
        if (args == null || args.length == 0) {
            System.err.println("usage: HealthCenterThreadDumpAnalyzer $ {HCDFILE}\n");
            return;
        }

        File file = new File(args[0]);
        message("Loading " + file.getAbsolutePath());

        HealthCenter hc = HealthCenterFactory.connect(file);
        message("Getting all thread dumps");

        ThreadsData threadsData = hc.getThreadsData();
        HashMap<Long, ThreadData[]> threadDumps =
                threadsData.getAllThreads();
        List<Entry<Long, ThreadData[]>> sortedThreadDumps = new ArrayList<>();
        for (Entry<Long, ThreadData[]> entry : threadDumps.entrySet()) {
            sortedThreadDumps.add(entry);
        }
        sortedThreadDumps.sort(new Comparator<Entry<Long, ThreadData[]>>() {
            @Override
            public int compare(Entry<Long, ThreadData[]> x, Entry<Long, ThreadData[]> y) {
                return x.getKey().compareTo(y.getKey());
            }
        });

        message("Processing all " + sortedThreadDumps.size() + " thread dumps");

        try (PrintWriter out = new PrintWriter(new File(file.getParentFile(),

for (Entry<Long, ThreadData[]> threadDump : sortedThreadDumps) {
    out.println(threadDumpDate.format(new Date(threadDump.getKey())));
    out.println("Full thread dump Java:");
    out.println();

    ThreadData[] threads = threadDump.getValue();
    Map<String, String> contendedMonitorOwners = new HashMap<>();
    for (int i = 0; i < threads.length; i++) {
        ThreadData thread = threads[i];
        if (thread.getContendedMonitor() != null && !thread.getContendedMonitor().isEmpty() && thread.getContendedMonitorOwner() != null && !thread.getContendedMonitorOwner().isEmpty()) {
            contendedMonitorOwners.put(thread.getContendedMonitorOwner(), thread.getContendedMonitor());
        }
    }

    for (int i = 0; i < threads.length; i++) {
        ThreadData thread = threads[i];
        String threadName = thread.getName();
        String nid = String.format("%08x", threadName.hashCode());
        nid = "0" + nid.substring(1);
        out.println("" + threadName + " ") + i + " daemon prio=9 os_prio=0 tid=0x" + String.format("%016x", threadName.hashCode()) + " nid=0x" + nid + " " + getThreadDumpThreadState(thread.getState()) + " [0x0000000000000000]";
        out.println(" java.lang.Thread.State: " + thread.getState());
        StackTraceElement[] threadStack = thread.getStackTrace();
        if (threadStack != null) {
            boolean firstFrame = true;
            for (StackTraceElement frame : threadStack) {
                out.println(" at " + frame);
                if (firstFrame) {
                    firstFrame = false;
                    if (thread.getContendedMonitor() != null && !thread.getContendedMonitor().isEmpty()) {
                        String contendedMonitor = thread.getContendedMonitor();
                        writeContendedMonitor(out, contendedMonitor, true);
                    }
                }
            }
        }
    }
}

for (MonitorInfo monitor : thread.getLockedMonitors()) {
    out.println(" - locked " + monitor);
throw new IllegalStateException("Not implemented " +
    monitor);
}

for (LockInfo lock : thread.getLockedSynchronizers()) {
    out.println("   - locked " + lock);
    throw new IllegalStateException("Not implemented " + lock);
}

for (String ownedMonitor : thread.getOwnedMonitors()) {
    int space = ownedMonitor.indexOf(' ');
    if (space != -1) {
        ownedMonitor = ownedMonitor.substring(0, space);
    }
    out.println("   - locked <0x" + String.format("%016x",
        ownedMonitor.hashCode()) + "> (a "
        + ownedMonitor + ")");
}

String contendedMonitor =
    contendedMonitorOwners.get(threadName);
if (contendedMonitor != null) {
    writeContendedMonitor(out, contendedMonitor, false);
}

out.println();
}

out.println();
}

message("Finished processing");
System.exit(0);

private static void writeContendedMonitor(PrintWriter out, String
    contendedMonitor, boolean waiting) {
    String className = contendedMonitor.substring(0,
        contendedMonitor.indexOf('@'));
    String identityHashCode =
        contendedMonitor.substring(contendedMonitor.indexOf('@') + 1);
    out.println("   - " + (waiting ? "waiting to lock" : "locked") + "
<0x" + String.format("%016x", Long.parseLong(identityHashCode, 16)) + ">
    (a " + className + ")");
}

public static String getThreadDumpThreadState(State threadState) {
    switch (threadState) {
    case BLOCKED:
        return "waiting for monitor entry";
    case NEW:
        return "new";
    case Runnable:
        return "Runnable";
    case TERMINATED:
        return "terminated";
    case TIMED_WAITING:
        return "timed waiting";
    case UNMAPPED:
        return "unmapped";
    case UNMAPPED_BLOCKED:
        return "unmapped blocked";
    default:
        return "unknown";
    }
}

public static String getThreadDumpThreadState() {
    switch (threadState) {
    case BLOCKED:
        return "waiting for monitor entry";
    case NEW:
        return "new";
    case Runnable:
        return "Runnable";
    case TERMINATED:
        return "terminated";
    case TIMED_WAITING:
        return "timed waiting";
    case UNMAPPED:
        return "unmapped";
    case UNMAPPED_BLOCKED:
        return "unmapped blocked";
    default:
        return "unknown";
    }
}

private static void writeContendedMonitor(PrintWriter out, String
    contendedMonitor, boolean waiting) {
    String className = contendedMonitor.substring(0,
        contendedMonitor.indexOf('@'));
    String identityHashCode =
        contendedMonitor.substring(contendedMonitor.indexOf('@') + 1);
    out.println("   - " + (waiting ? "waiting to lock" : "locked") + "
<0x" + String.format("%016x", Long.parseLong(identityHashCode, 16)) + ">
    (a " + className + ")");
}
return "Runnable";
case TERMINATED:
    return "terminate";
case TIMED_WAITING:
    return "waiting on condition";
case WAITING:
    return "waiting on condition";
default:
    throw new IllegalStateException("Not implemented for " + threadState);
}
}

public static void message(String message) {
    System.out.println("[" + new Date() + "] " + message);
}

By default, Health Center captures full stacks. If this appears to be a performance impact, you can limit this with -Dcom.ibm.java.diagnostics.healthcenter.thread.stack.depth=${MAXDEPTH}
The default thread stack collection interval is 30 seconds. This can be changed with
-Dcom.ibm.java.diagnostics.healthcenter.thread.collection.interval=${SECONDS}
To disable collection of thread stacks: -Dcom.ibm.diagnostics.healthcenter.data.threads=off


Extracting JVM Trace
1. Unzip the hcd
2. Remove the first 12 bytes of the trace file: dd bs=12 skip=1 if=trace of=trace.cut
3. Format the trace file: java com.ibm.jvm.TraceFormat trace.cut

Updating to the latest agent
1. Stop the application servers and node agents (and DMGR if on that node).
2. Download the latest agent and follow the installation instructions here:
3. If WAS runs under a non-root user, make sure to chown properly.
4. Re-start the application servers.

On AIX, if you had previously run Health Center, even if you stop all JVMs, you will probably see this error extracting libhealthcenter.so:

tar: 0511-188 Cannot create ./jre/lib/ppc64/libhealthcenter.so: Cannot open or remove a file containing a running program.

By default, AIX will keep shared libraries in memory even after all JVMs referencing that library have stopped. To remove the shared library from memory, you may either reboot the box or more simply, run the slibclean command (see also genkld and genld). This should be safe to run because it only affects
shared libraries that have no current load or use counts:

"The slibclean command unloads all object files with load and use counts of 0. It can also be used to remove object files that are no longer used from both the shared library region and in the shared library and kernel text regions by removing object files that are no longer required." (http://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.cmds5/slibclean.htm)

Now you should be able to overwrite libhealthcenter.so

You may also find that healthcenter.jar has open file handles (e.g. lsof) in Java processes even if healthcenter was not enabled. This is because healthcenter.jar is in the "ext" JRE directory which is searched as part of some classpath operations. If you take a system dump, you will find a java.util.jar.JarFile object with a name field that includes healthcenter.jar, and this JarFile object probably has a native file handle open (although you will not find a java.io.File object with that path). In theory, it should be safe to overwrite healthcenter.jar even if running processes have open file handles to it because the JAR file will not be read by those JVMs that do not have healthcenter enabled.

It is possible to update to the latest agent without modifying the binaries in the WAS folder:

1. Extract the agent ZIP into any directory; for example, /work/healthcenter/agent/
2. Take a javacore of the running target server and find the last value of -Djava.ext.dirs (note that there may be multiple instances, so always take the last value). For example:
   -Djava.ext.dirs=/work/local/was85/tivoli/tam:/work/local/was85/java/jre/lib/ext
3. Prepend the path to the ext folder under the expanded HealthCenter agent directory to
   -Djava.ext.dirs. For example:
   -Djava.ext.dirs=/work/healthcenter/agent/jre/lib/ext:/work/local/was85/tivoli/tam:/work/local/was85/java/jre/lib/ext
4. Append this parameter as well as the following parameters (replacing the path to the HealthCenter agent) to the generic JVM arguments:
   -Djava.ext.dirs=/work/healthcenter/agent/jre/lib/ext:/work/local/was85/tivoli/tam:/work/local/was85/java/jre/lib/ext -agentpath:/work/healthcenter/agent/jre/bin/libhealthcenter.so
   -Dcom.ibm.java.diagnostics.healthcenter.agent.properties.file=/work/healthcenter/agent/jre/lib/healthcenter.properties
5. Append this parameter to the "Classpath" textbox on the same page as the generic JVM arguments (replacing the path to the HealthCenter agent):
   /work/healthcenter/agent/jre/lib/ext/healthcenter.jar
6. Add the necessary HealthCenter arguments described above to enable it.
7. Restart the JVM.

**Low mode**

-Xhealthcenter:level=low disables method profiling since this has the highest overhead and creates the most data. This would be useful if you wanted something else from health center (e.g. garbage collection, native memory, etc.) with less overhead.

Low cannot be combined with headless (e.g. -Xhealthcenter:level=low,level=headless), so the way to do it is to use headless mode and then: In jre/lib/ext there is a file called healthcenter.jar. If you unpack that you will find a file called TRACESourceConfiguration.properties and this is what defines which data is switched on by Trace. When we run in low mode, we turn off one of the profiling trace points. You can do this manually by editing this file and finding the entry "j9jit.16=on" and then changing it to
"j9jit.16=off". If you repackage the jar up you should find that the amount of trace generated is a lot less (but you won't get method profiling).

**Hexadecimal Method Names**

Sometimes you may see addresses (0x0123ABCD) instead of method names. This usually occurs for methods loaded very early in the JVM such as classloading methods.

This issue has been resolved with Java >= 6 SR11 although it requires the latest HC agent.

**Health Center Details**

For general questions, send an email to javatool@uk.ibm.com.

In one of the largest customer production situations, health center wrote about 5GB per hour of data to the filesystem.

**Apache JMeter**

Apache JMeter ([http://jmeter.apache.org/](http://jmeter.apache.org/)) has a bit of a learning curve but generally has all the features needed to do performance testing. Writing and maintaining realistic test suites can be time consuming, particularly because even minor changes to an application can break the test flow and assumptions. Nevertheless, it is critical to have realistic testing. You can have different tiers of tests, from simple smoke tests to incredibly realistic user flows, with the latter being more brittle.

Download the JMeter binary ([http://jmeter.apache.org/download_jmeter.cgi](http://jmeter.apache.org/download_jmeter.cgi)), unzip, change directory to bin, and run jmeter. You will start with a blank test plan and workbench. In general, you should do most of your work in the test plan and the workbench is only used for some copy/paste operations. Right click on the test plan and use the context menus to build it. Here are some general tips:

- As you change fields and navigate, the changes you make are persistent within the GUI; however, you should save your plan and periodically re-save as you make changes.
- Try to use variables as much as possible so that your test is more flexible.

See a sample JMeter script at [https://raw.githubusercontent.com/kgibm/problemdetermination/master/scripts/jmeter/sample_test_plan.jmx](https://raw.githubusercontent.com/kgibm/problemdetermination/master/scripts/jmeter/sample_test_plan.jmx), a screenshot of which is below. Here are the highlights:

- A "User Defined Variables" configuration element defines some global variables such as the scheme, host, port, number of threads, etc.
- An "HTTP Request Defaults" configuration element defines the default parameters of the HTTP client. In particular, note that "Retrieve All Embedded Resources" and "Use concurrent pool" are checked to instruct the client to retrieve things such as images, CSS, and JS resources from resulting HTML files to more closely mimic real world behavior.
- An "HTTP Header Manager" configuration element with a header name of "Authorization" and a value of "Basic ...
  shows how to add an HTTP header to perform basic authorization on every request. Notice that the element is grayed out, signifying that the element is disabled. To enable it, right click and click Enable or Toggle. This technique is often useful to quickly change tests.
- A "Poisson Random Timer" timer element pauses each thread for a random period of time between requests with most times occurring near the specified value in the configuration.
• A "Thread Group" threads element that will perform the actual HTTP requests with a certain concurrency and for a certain number of iterations.
  • An "HTTP Cookie manager" configuration element that will stores cookies for each thread.
  • An "HTTP Request" sampler element that will do the actual HTTP request. Since we've set up HTTP Request Defaults above, we only need to change what's unique to this request, in the first example just the path /
  • A "Response Assertion" assertion element that will fail the request if it doesn't see the specified value in the response. It is useful to add these to all responses to ensure that there are no functional errors in the application.
  • An "XPath Extractor" post processor element which will extract content from the response into variables for use in subsequent requests. We check "Use tidy (tolerant parser)" because most HTML is not well formed XML. We set the reference name to the variable that we want to hold the extraction, and the XPath query to perform the extraction. Other useful post processors are the regular expression extractor.
• An "HTTP Request" sampler element that will do an HTTP request to the contents of the variable that we extracted from the previous response.
• A "Summary Report" listener element that will provide basic statistics on the test results.
• A "Graph Results" listener element that will provide the same statistics as the summary report in graph form over time.
• A "View Results Tree" listener element that will provide the full request and response of every sample. This is useful during test design and should be toggled off otherwise.
• Make a habit to change the "Name" of each element to describe what it's doing. The name will be reflected in the tree on the left.
• To start the test, click the simple green arrow.
• As the test is running, the number of threads executing is in the top right corner. You can also click any of the listener elements to see a live view of the statistics, graph, or results tree.
• To stop the test, click the shutdown red X button (the stop button terminates threads and should be avoided).
• After stopping a test, you may want to clear the previous results before starting a new iteration. Click the clear all brooms icon to reset the listeners.
Wireshark

Wireshark is an open source program to perform analysis on capture packets:
https://www.wireshark.org/. Wireshark supports the packet formats of most operating systems.

- A frame is basically a "packet."
- A conversation is the set of packets between two endpoints.
- An endpoint is a logical endpoint of a protocol or network layer. For most purposes, focusing on an IP endpoint, i.e. an IP address.
- Following a stream means extracting the subset of a conversation, from the point of view of an application. For most purposes, focusing on a TCP stream, i.e. SYN -> SYN/ACK -> ACK -> ... -> FIN -> FIN/ACK -> ACK
- There is no way with a single capture to know how long it took for the packet to be transmitted. This requires a correlated packet capture on the other side where the packet was sent from/to.
- Timestamp of packet is:
  - For an incoming packet, the timestamp is when the capture mechanism is handed the packet from its way from the NIC to the client. This would include any transition time over the NIC.
  - For an outgoing packet, the timestamp is when the capture mechanism is handed the packet from its way from the client to the NIC, before it hits the NIC.
- Split a capture: editcap

It is not unusual for bugs in switch firmware to sometimes cause negative response times. You are looking for things like frequent packet retransmissions or packet loss. Be careful when using Wireshark because it frequently marks issues in the results that are not necessarily issues so it takes a lot of expertise when doing packet analysis and comparing what you see with the search results.

Common Things to do

- Statistics > Summary
  - Note "Between first and last packet" and "Bytes."
- Analyze > Expert Info > Filter to Error+Warn only
  - See if there is anything notable.
- Statistics > Protocol Hierarchy
  - Note percentages of packet types.
- Statistics > IO Graphs > Change "Y Axis" to "Bytes/Tick"
  - Note the throughput over time.

Lua Scripts

The command line version of Wireshark, tshark, supports Lua scripts to perform automated analysis. For example, here is a script that checks for common TCP anomalies and long delays:

Description: https://wiki.wireshark.org/Lua

```
$ cat file.lua
print("hello world!")
$ tshark -X lua_script:file.lua
```
Finding problems or delays in the network

- Ensure all suspect servers have synchronized clocks (NTP, etc.).
- Run a few minutes of pings and note the average latency between servers.
- Capture network trace from both servers. The network trace should include both incoming and outgoing packets, to and from the other servers. If they are unidirectional, it turns out merging network traces from two different servers is very dangerous. Basically, a lot of information such as DUP ACKs, etc., is inferred from the sequence of packets, and combining multiple systems' traces can have unintended consequences. In my case, it actually generated TCP retransmits when they did not exist.
- In Wireshark
  - Select View -> Time/Display Format -> Date and Time of Day
  - Also, add a column for "Delta time (Displayed)"
- Open the first capture. Basically, we will use the frame.time_delta_displayed column we added above to find the delays. However, the column is non-sensical with interleaved TCP streams. So what you have to do is basically filter by each stream (using either "Follow TCP stream" on a suspect packet, or finding all unique src/destination port combos, etc.).
- Once you have a single TCP stream, then the frame.time_delta_displayed is the time between packets on that stream. Sort by this column, descending.
- If there are any big delays (larger than the max latency), then note a few of those frames. Re-sort by frame # and see what was happening right before that frame.
- Note that a TCP "stream" within a TCP "conversation" is just the unique combination of Source IP, Source Port, Destination IP, and Destination Port. TCP is multiplexing so multiple sockets can be opened between two IPs. What this also means is that if the communication stream is persistent (e.g. connection pooling), then the stream is never torn down (FIN -> FIN/ACK -> ACK), so there may be large gaps between packets on that stream which may just be the time between two units of work on that stream.
- Note any errors within Wireshark. Research each error carefully as some may be benign (e.g. TCP Checksum Offloading).
- Do the same thing on the other end and compare the time stamps. Remember that the timestamps in a packet capture are the time at which the capturer is handed the packet. For an outgoing packet, this occurs before the packet hits the NIC. For an incoming packet, this occurs after the packet has been processed by the NIC and handed off to the kernel.
- Any time difference between when server A receives the response from server B (from server A's packet capture), and when server B sends the pack to server B (from server B's packet capture) would be the latency. Any other time would mean the time taken to process on server B.
- Also, to find any clock difference, pick any TCP conversation handshake. The SYN/ACK must come after the SYN and before the ACK, so you can shift one packet capture or the other (using editcap -t) to line up with the other. For example, when server B is sending the SYN/ACK and it is behind the SYN, use the following to time shift server B's packet capture: (((ACK - SYN) / 2) + SYN) - (SYNACK)
Finding gaps within an IP conversation in a network capture

- [https://www.wireshark.org/docs/dref/f/frame.html](https://www.wireshark.org/docs/dref/f/frame.html)
  - frame.time_delta -> Time delta between the current packet and the previous packet in the capture (regardless of any display filters).
  - frame.time_delta_displayed -> Time delta between the current packet and the previous packet in the current display.
  - frame.time_relative -> Time delta between the current packet and the first packet in the capture (regardless of any display filters), or if there is a time reference, that reference time.

To find gaps within an IP conversation:

- First add frame.time_delta_displayed column: Edit -> Preferences -> User Interface -> Columns -> Add -> Field Type = Delta Time Displayed.
- To find gaps, apply some logical grouping to the packets so that they are all related, e.g. right click on the SYN of the incoming/outgoing packet and click "Follow TCP Stream." Close the window that pops up and now Wireshark is filtered to that particular tcp stream (e.g. "tcp.stream eq 5"). (This could also be done just with the conversation, not just the stream).
- The Delta Time Displayed is the delta time between that packet and the previous packet in that stream -- i.e. the gap between packets in that conversation.
- Another interesting thing to do is to colorize large differences in frame.time_delta_displayed: View -> Coloring Rules -> New -> Filter: frame.time_delta_displayed >= .1

TCP Checksum Offloading Errors

TCP Checksum Offloading: Checksum offloading is when the OS network driver does not perform a checksum, but instead fills the checksum with 0 or garbage, and then 'offloads' the checksum processing to the physical NIC card which then itself does the checksum and puts it in the packet before sending it off. Thus a capture will get a garbage checksum. Checksum offloading errors within Wireshark are only benign if the packets are outgoing. Two ways to avoid are: 1) turn off the OS checksum offloading (not always possible or simple, and could significantly impact performance), or 2) turn off checksum validation in Wireshark. For #2: Edit -> Preferences -> Protocols -> TCP -> Uncheck "Check the validity of the TCP checksum when possible."

- [https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html](https://www.wireshark.org/docs/wsug_html_chunked/ChAdvChecksums.html)
- [https://www.wireshark.org/faq.html#q11.1](https://www.wireshark.org/faq.html#q11.1)
- [https://wiki.wireshark.org/TCP_Checksum_Verification](https://wiki.wireshark.org/TCP_Checksum_Verification)
- [https://wiki.wireshark.org/TCP_Reassembly](https://wiki.wireshark.org/TCP_Reassembly)

Decrypt SSL/TLS Traffic

By default, WAS uses PKCS#12 or p12 files for the SSL key stores with the password of WebAS; for example, $WAS/profiles/$DMGR/etc/key.p12 file. If you are a customer providing a p12 file to IBM (or anyone, really), check with your security team on rules and procedures. There are also ways to export just the RSA private key part out of the p12 file without a password.

Start Wireshark and go to Edit > Preferences > Protocols > SSL. Enter:
RSA keys list: CLIENT_IP, SERVER_SSL_PORT, http, PATH_TO_P12_FILE, P12_PASSWORD
SSL debug file: PATH_TO_DEBUG_FILE

For example:
RSA keys list: 127.0.0.1, 9094, http, /tmp/wire/key.p12, WebAS
SSL debug file: /tmp/wire/wireshark.out

For native IBM components that use *.kdb keystores, use Ikeyman or gskcapicmd to export the private keys to a new PKCS12 file:
/opt/IHS/bin/gskcapicmd -cert -export -db key.kdb -pw XXX -label XXX -type cms -target cert.pfx -target_type pkcs12

If a frame on a stream says "Encrypted Handshake Message," then something went wrong. Looking at the SSL debug file, search for CIPHER:
$ grep CIPHER wireshark.out
dissect_ssl3_hnd_srv_hello found CIPHER 0x0033 -> state 0x17

Looking at the IANA.org list, we can see that this cipher is TLS_DHE_RSA_WITH_AES_128_CBC_SHA (http://www.iana.org/assignments/tls-parameters/tls-parameters.xml).

Wireshark does not currently support DHE without a log of identifiers. There are three solutions to this: 1) change the clients to not use DHE ciphers, 2) change WAS to not use DHE ciphers, or 3) log the session identifiers and pass them to Wireshark. Check with your security team before making these types of changes.

To remove DHE from available ciphers in WAS, go to SSL certificate and key management > SSL configurations > $SSL_SETTINGS > Quality of protection (QoP) settings. Under Cipher suite settings, select any ciphers in the Selected ciphers listbox that have DHE in them, and click << Remove. Click OK, save, and synchronize.

It's important to note that the SSL handshake must be captured for decryption to work, so if the packet captures starts and the user's browser re-uses a previous connection, the user will need to restart their browser or wait for that connection to naturally time out.

tshark -o "ssl.desegment_ssl_records: TRUE" -o "ssl.desegment_ssl_application_data: TRUE" \ -o "ssl.keys_list: 127.0.0.1,9094,http,key.pem" -o "ssl.debug_file: wiresharkssl.out" \ -r $FILE.pcap -X lua_script:printhttp.lua

Some browsers support an SSL session key log file. For example, using the SSLKEYLOGFILE environment variable: https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Key_Log_Format

Ports and Heuristics
In general, Wireshark uses two mechanisms to decide whether a protocol should dissect packets: ports and heuristics.
Ports are usually specified on a per-protocol basis under Edit > Preferences > Protocols. For example, if HTTP traffic is running on a "non-standard" port, you may add the additional ports to the HTTP protocol.

Heuristics are optionally implemented by protocols to guess that a stream is of the protocol's type. Some protocols do not expose an option to disable their heuristic, in which case the protocol may be disabled under Analyze > Enabled Protocols.

**Working with Wireshark Source**

Example building on Linux with older GTK:

```
$ ./autogen.sh
$ ./configure --with-gtk2
$ make
$ ./wireshark-gtk
```

Launching Wireshark in GDB:

```
$ libtool --mode=execute gdb -ex=run -ex=quit ./wireshark-gtk
$ libtool --mode=execute gdb -ex=run -ex=quit --args ./wireshark-gtk
file.pcap
$ libtool --mode=execute gdb -ex=run -ex=quit --args ./wireshark-gtk -R
'tcp.stream == 3' file.pcap
```

Abort on a dissector bug:

```
$ export WIRESHARK_ABORT_ON_DISSECTOR_BUG=1
```

**Custom Dissector**

For a template, see doc/packet-PROTOABBREV.c. To compile into Wireshark, add the file into epan/dissectors, and add its name to DISSECTOR_SRC in epan/dissectors/Makefile.common. See doc/README.developer and doc/README.dissector.

**IBM Interactive Diagnostic Data Explorer**

**Installation**

This tool is one among the many Problem Determination Tools that comes free with ISA 5.0. For installation instructions, see the IBM Support Assistant chapter.

**Useful commands**

The "!threads flags" extension is useful to show which thread grabbed exclusive access. If the thread that is performing the dump does not have exclusive access, then the dump may be too corrupt to read. For example, another thread may have just acquired exclusive access to perform a garbage collection.

Show the various extents of the Java heap:

```
> info heap *
```
IBM Support Assistant (ISA)

This cookbook describes how to use major tools through Eclipse rather than through ISA; however, most of the functionality is the same. If you don't plan to use ISA, you may skip this chapter.

The IBM Support Assistant Team Server version 5 is a free tool that provides a standalone or hosted, web-based problem determination environment: [http://www-01.ibm.com/software/support/isa/teamserver.html](http://www-01.ibm.com/software/support/isa/teamserver.html)

For those familiar with ISA version 4, ISA 5 is quite different. Fundamentally, both are free platforms that can install and run tools; however, ISA 4 is a GUI thick client and ISA 5 is web focused. In general, we recommend customers install ISA 5 on a very powerful server which is then accessed by team members through a web page (any tools that must be run locally are downloaded and launched through Java Web Start); however, you can also download and install ISA 5 on your local machine and open your browser to [http://localhost:10911/isa5](http://localhost:10911/isa5)

ISA 5 includes a very powerful log analysis engine called Phase 1 Problem Determination (P1PD) that finds common warnings and errors and proposes various solutions through the "Scan Logs" button: [https://www.ibm.com/support/knowledgecenter/en/SSLLVC_5.0.0/com.ibm.isa.help.doc/html/overview/problemmdetermination.html](https://www.ibm.com/support/knowledgecenter/en/SSLLVC_5.0.0/com.ibm.isa.help.doc/html/overview/problemmdetermination.html)

**Installation**

There are many different ways to install and use ISA5:

2. Use IBM Installation Manager with the following repository: [https://www.ibm.com/software/repositorymanager/com.ibm.isa.offerings.v5](https://www.ibm.com/software/repositorymanager/com.ibm.isa.offerings.v5)
3. Same as above but expand the main ISA tool offering and choose the "EAR" option to download EAR files which can then be installed into separate installations of WAS.

**Starting ISA5**

1. Run the start_isa.bat or start_isa.sh script in the ISA5 installation directory. The script will start three different WAS Liberty Java processes, one for the overall ISA5 tool itself, another for the MAT Web Edition tool, and the third for the IDDE Web edition tool. When the script prints, "Press ENTER to finish..." you may press ENTER and the start script will finish. Pressing ENTER
will not stop the servers (there's a separate stop_isa.bat/stop_isa.sh script for that). So, feel free to press ENTER; nothing will happen and you'll get your terminal back or close the window.

```
[user1@20:23:14 ~]$ /opt/IBM/ISA/ISA5/start_isa.sh
```

NOTICE:
IBM Support Assistant is being started without Administrator privileges, which were used to perform the initial installation. Restart the server with Administrator privileges if you would like to install and update problem determination tools through the IBM Support Assistant Administration panel. Otherwise, use IBM Installation Manager to install and update problem determination tools.

Now starting IBM Support Assistant Team Server.
System resources and system load may affect the time required to start the application. Please be patient...

Starting server isa.
Server isa started with process ID 3571.

Starting server com.ibm.java.web.idde.
Server com.ibm.java.web.idde started with process ID 4059.

Starting server com.ibm.java.web.memoryanalyzer.
Server com.ibm.java.web.memoryanalyzer started with process ID 4190.

IBM Support Assistant is ready to run.
Open a browser to:

Press ENTER to finish...

2. Open a browser and go to http://localhost:10911/isa5 (replace `localhost` with the target hostname if running remotely)
3. You may create cases and upload and interact with files, or you may immediately run tools through the Tools tab. For example:
Java Web Start Tools

All of the tools with "[Desktop]" in the name are GUI tools launched through Java Web Start. When first launching each tool, you may receive a warning such as the following which you can click Continue through:

There may be long delays while launching tools using JWS. On some versions of Linux, there is a known issue, seemingly with SWT-based applications such as HealthCenter, where the program becomes hung and never launches. It appears this is a race condition in SWT and it is usually worked around by enabling the Java Console in the Java ControlPanel application of the Java on the path.
Specifying the Java Maximum Heap Size

Most of the Desktop JWS tools allow you to specify the maximum Java heap size in a small browser popup overlay when launching the tool:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>minheap</td>
<td>Minimum Java heap size (MB) or blank to use system default</td>
<td>128</td>
</tr>
<tr>
<td>maxheap</td>
<td>Maximum Java heap size (MB) or blank to use system default</td>
<td>512</td>
</tr>
</tbody>
</table>

R Project

This cookbook may generate graphs and calculate statistics using the free R project: http://www.r-project.org/. You may use other techniques or programs to do the same; otherwise, please install R and the following extensions and become familiar with its basic usage.

R is designed to work on Unix, Windows, and Mac. R is normally distributed with operating system package managers (e.g. "yum install R" with epel.repo enabled=1 in RHEL), or you can download binary or source packages from https://cran.rstudio.com/.

To run R from the command line, simply type R and you'll be in a read-evaluate-print-loop (REPL). Some basic commands you'll need:

- q() to quit (usually type 'n' to discard the workspace)
- ?CMD to get help on CMD

We'll be using some external packages so the first time you use R, you'll need to install them:

```r
> install.packages(c("xts", "xtsExtra", "zoo", "txtplot"),
repos=c("http://cran.us.r-project.org","http://R-Forge.R-project.org"))
```

R has its own package management system and this will download the specified third party packages from the web.

Install Package from Source

```r
> install.packages("http://download.r-forge.r-project.org/src/contrib/xtsExtra_0.0-1.tar.gz", repos=NULL, type="source")
```

Another example:

```
$ svn checkout --revision 850 svn://svn.r-forge.r-project.org/svnroot/xts/
$ R
> install.packages("xts/pkg/xts", repos=NULL, type="source")
> install.packages("xts/pkg/xtsExtra", repos=NULL, type="source")
```
Graphing CSV Data

An example script is provided which graphs arbitrary time series data in a comma separated value (CSV) file using plot.xts. The script expects the first column to be a time column in the following format: YYYY-MM-DD HH:MM:SS

For example, with the following CSV file:

```
Time, Lines, Bytes
2014-12-04 13:32:00, 1043, 12020944
2014-12-04 13:33:00, 212, 2737326
2014-12-04 13:34:00, 604, 13982275
2014-12-04 13:35:00, 734, 190323333
2014-12-04 13:36:00, 1256, 126198301
2014-12-04 13:37:00, 587, 72622048
2014-12-04 13:38:00, 1777, 237571451
```

Optionally export environment variables to control the output:

```
$ export INPUT_TITLE="Data"
$ export INPUT_PNGWIDTH=600
$ export INPUT_PNGHEIGHT=300
$ export TZ=UTC
```

Run the example script with the input file:

```
$ git clone https://github.com/kgibm/problemdetermination
$ R --silent --no-save -f problemdetermination/scripts/r/graphcsv.r < test.csv
```

The script generates a PNG file in the same directory:

![Data (Timezone UTC)](image)

Package Versions

Display loaded package versions:
Test Graphing

Test graphing with the following set of commands:

```r
$ R
library(zoo)
library(xts)
library(xtsExtra)
sessionInfo()
timezone = "UTC"
Sys.setenv(TZ=timezone)
sampleData = "Time (UTC),CPU,Runqueue,Blocked,MemoryFree,PageIns,ContextSwitches,Wait,Steal
2014-10-15 16:12:11,20,0,0,12222172,0,2549,0,0
2014-10-15 16:12:12,27,1,0,12220732,0,3619,0,0
2014-10-15 16:12:13,30,0,0,12220212,0,2316,0,0"
data = as.xts(read.zoo(text=sampleData, format="%Y-%m-%d %H:%M:%S",
header=TRUE, sep="","", tz=timezone))
plot.xts(data, main="Title", minor.ticks=FALSE, yax.loc="left",
auto.grid=TRUE, nc=2)
```

Common Use Case

```r
> options(scipen = 999)
> x = read.csv("tcpdump.pcap.csv")
> x = na.omit(x[,"tcp.analysis.ack_rtt"])
> summary(x)
  Min.  1st Qu.   Median     Mean  3rd Qu.     Max.  
0.00000020 0.00000050 0.00000070 0.00001850 0.0002290 0.12220000
> sum(x)
[1] 58.69276
> length(x)
[1] 306702
> quantile(x, 0.99)
 99%
0.000388
> plot(density(x[ x < quantile(x, 0.99)]))
```
Web Servers

Web Servers Recipe

1. The maximum concurrency variables (e.g. MaxClients for IHS and optionally/rarely MaxConnections for the WAS plugin) are the key tuning variables. Ensure MaxClients is not saturated through tools such as mpmstats or mod_status, while at the same time ensuring that the backend server resources (e.g. CPU, network) are not saturated (this can be done by scaling up the backend, sizing thread pools to queue, optimizing the backend to be faster, or with plugin MaxConnections).

2. Use WAS Traditional clusters or WAS Liberty collectives to scale out work over multiple systems and processes, both for fault tolerance and increasing capacity.

3. Clusters of web servers are often used with IP sprayers or caching proxies balancing to the web servers. Ensure that such IP sprayers are doing "sticky SSL" balancing so that SSL Session ID reuse percentage is higher.

4. Load should be balanced evenly into the web servers and back out to the application servers. Compare access log hit rates for the former, and use WAS plugin STATS trace to verify the latter.

5. Review snapshots of thread activity to find any bottlenecks. For example, increase the frequency of mpmstats and review the state of the largest number of threads.

6. Review the keep alive timeout. The ideal value is where server resources (e.g. CPU, network) are not saturated, maximum concurrency is not saturated, and the average number of keepalive requests has peaked (use mpmstats or mod_status).

7. Check the access logs for HTTP response codes (e.g. %s for IHS) >= 400.

8. Check the access logs for long response times (e.g. %D for IHS).

9. Review access and error logs for any errors, warnings, or high volumes of messages.

10. Use WAS plugin DEBUG or TRACE logging to dive deeper into unusual requests such as slow requests, requests with errors, etc. Use an automated script for this analysis: https://github.com/covener/plugin-tools/blob/master/scanplugin.pl

11. Fine-tuning of SSL ciphers or other MPM configuration directives is unlikely to have a big impact.
Also review the operating systems chapter.

**General**

"Web servers like IBM HTTP Server are often used in front of WebSphere Application Server deployments to handle static content or to provide workload management (WLM) capabilities. In versions of the WebSphere Application Server prior to V6, Web servers were also needed to effectively handle thousands of incoming client connections, due to the one-to-one mapping between client connections and Web container threads... In WebSphere Application Server V6 and later, this is no longer required with the introduction of NIO and AIO. For environments that use Web servers, the Web server instances should be placed on dedicated systems separate from the WebSphere Application Server instances. If a Web server is collocated on a system with a WebSphere Application Server instance, they will effectively share valuable processor resources, reducing overall throughput for the configuration."


Locating the web server on a different machine from the application servers may cause a significant throughput improvement. In one benchmark, 27%


**IBM HTTP Server**

The IBM HTTP Server is based on the open source Apache httpd code with IBM enhancements. General performance tuning guidelines:


Although IHS is supported on Windows 64-bit, it is only built as a 32-bit executable. So in all cases on Windows, IHS is limited to a 32-bit address space. IHS on Windows also only supports a single child process

[https://www.ibm.com/support/knowledgcenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rprf_pl ugin.html](https://www.ibm.com/support/knowledgcenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rprf_plugin.html). IHS on Windows is not /LARGEADDRESSAWARE, so it cannot utilize the extra space afforded by the /3GB switch. After APAR PI04922 Windows services created with the httpd-la.exe binary are large address aware (which does not depend on /3GB boot time option): [http://www-01.ibm.com/support/docview.wss?uid=swg1PI04922](http://www-01.ibm.com/support/docview.wss?uid=swg1PI04922)

Note also on Windows that there is no MaxClients. It is set implicitly to ThreadsPerChild.

Consider using mod_disk_cache instead of the in-memory cache:


**Multi-Processing Modules (MPM)**

Requests are handled by configurable multi-processing modules (MPMs)


- worker: This is the default, multi-threaded and optionally multi-process MPM.


- event: Built on top of worker and designed to utilize more asynchronous operating system APIs
• prefork: A single thread/process for each request. Not recommended. Generally used for unthread safe or legacy code.

This is the default configuration on distributed platforms other than Windows:

```
# ThreadLimit: maximum setting of ThreadsPerChild
# ServerLimit: maximum setting of StartServers
# StartServers: initial number of server processes to start
# MaxClients: maximum number of simultaneous client connections
# MinSpareThreads: minimum number of worker threads which are kept spare
# MaxSpareThreads: maximum number of worker threads which are kept spare
# ThreadsPerChild: constant number of worker threads in each server process
# MaxRequestsPerChild: maximum number of requests a server process serves

<IfModule worker.c>
  ThreadLimit         25
  ServerLimit         64
  StartServers         1
  MaxClients         600
  MinSpareThreads     25
  MaxSpareThreads     75
  ThreadsPerChild     25
  MaxRequestsPerChild  0
</IfModule worker.c>
```

Out of the box, IBM HTTP Server supports a maximum of 600 concurrent connections. Performance will suffer if load dictates more concurrent connections, as incoming requests will be queued up by the host operating system...

First and foremost, you must determine the maximum number of simultaneous connections required for this Web server. Using mod_status or mod_mpmstats (available with ihsdiag) to display the active number of threads throughout the day will provide some starting data.

There are 3 critical aspects to MPM (Multi-processing Module) tuning in IBM HTTP Server.

1. Configuring the maximum number of simultaneous connections (MaxClients directive)
2. Configuring the maximum number of IBM HTTP Server child processes (ThreadsPerChild directive)
3. Less importantly, configuring the ramp-up and ramp-down of IBM HTTP Server child processes (MinSpareThreads, MaxSpareThreads, StartServers)

The first setting (MaxClients) has the largest immediate impact, but the latter 2 settings help tune IBM HTTP Server to accommodate per-process features in Apache modules, such as the WebSphere Application Server Web server plug-in.


This is the default configuration on Windows:

```
  ThreadLimit       600
```
ThreadsPerChild 600
MaxRequestsPerChild 0

In general, recommendations for a high performance, non-resource constrained environment:

- If using TLS, then ThreadsPerChild=100, decide on MaxClients, and then
  ServerLimit=MaxClients/ThreadsPerChild; otherwise, ThreadsPerChild=MaxClients and
  ServerLimit=1.
- StartServers=ServerLimit
- MinSpareThreads=MaxSpareThreads=MaxClients
- MaxRequestsPerChild=0
- Test the box at peak concurrent load (MaxClients); for example: ${IHS}/bin/ab -c
  {MaxClients} -n ${MaxClients*10} -i https://localhost/

Note that the message "Server reached MaxClients setting" in the error_log will only be shown once
per running worker process.

The WAS Plugin is loaded once per process, and thus counters such as MaxConnections are not
coordinated (http://www-01.ibm.com/support/docview.wss?uid=swg21167658). "For this reason it is
recommended that you use only a single web server child process with many threads." (http://www-
01.ibm.com/support/docview.wss?uid=swg21318463)

IBM HTTP Server typically uses multiple multithreaded processes for serving requests.
Specify the following values for the properties in the web server configuration file
(httpd.conf) to prevent the IBM HTTP Server from using more than one process for serving
requests.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerLimit</td>
<td>1</td>
</tr>
<tr>
<td>ThreadLimit</td>
<td>1024</td>
</tr>
<tr>
<td>StartServers</td>
<td>1</td>
</tr>
<tr>
<td>MaxClients</td>
<td>1024</td>
</tr>
<tr>
<td>MinSpareThreads</td>
<td>1</td>
</tr>
<tr>
<td>MaxSpareThreads</td>
<td>1024</td>
</tr>
<tr>
<td>ThreadsPerChild</td>
<td>1024</td>
</tr>
<tr>
<td>MaxRequestsPerChild</td>
<td>0</td>
</tr>
</tbody>
</table>

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc
/ae/rprf_plugin.html

Note that when TLS processing is enabled, there is some inter-process contention (buffers, etc.) so
more processes and less processes per threads may be faster:
http://publib.boulder.ibm.com/httpserv/ihsdiag/ihs_performance.html#Linux_Unix_ThreadsPerChild

**MinSpareThreads, MaxSpareThreads**

The MinSpareThreads and MaxSpareThreads options are used to reduce memory utilization during low
traffic volumes. Unless this is very important, set both of these equal to MaxClients to avoid time spent
destroying and creating threads.
**MaxRequestsPerChild**

The MaxRequestsPerChild option recycles a thread after it has processed the specified number of requests. Historically, this was used to prevent a leaking thread from using too much memory; however, it is generally recommended to set this to 0 and investigate any observed leaks.

**IBM HTTP Server for z/OS**


Consider using AsyncSockets=yes in httpd.conf ([http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/8dc00e9e9899e23886257100001fb274/$FILE/WP100719%20-%20Benefits%20of%20Asynch%20IO%20and%20PQ86769.pdf](http://www-03.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/8dc00e9e9899e23886257100001fb274/$FILE/WP100719%20-%20Benefits%20of%20Asynch%20IO%20and%20PQ86769.pdf))

**Access Log, LogFormat**

The access log is enabled by default and writes one line for every processed request into logs/access.log. The format of the line is controlled with the LogFormat directive in httpd.conf: [http://publib.boulder.ibm.com/httpserv/manual70/mod/mod_log_config.html](http://publib.boulder.ibm.com/httpserv/manual70/mod/mod_log_config.html)

The access log is defined with the CustomLog directive, for example:

```
CustomLog logs/access_log common
```

The last part (e.g. "common") is the name of the LogFormat to use. Here is the default "common" LogFormat:

```
LogFormat "%h %l %u %t "%r" %>s %b common
```

You can either modify this line or add a new LogFormat line with a new name and change the CustomLog to point to the new one.

We recommend adding at least %D to give the total response time (in microseconds).

```
LogFormat "%h %l %u %t "%r" %>s %b %D common
```

Here are some other commonly useful directives:

- Print the time taken to serve the request, in microseconds: %D
- Print the time taken to serve the request, in seconds: %T
- Print the contents of the cookie JSESSIONID in the request sent to the server (also includes the clone ID that the cookie wants the request to go back to): %{JSESSIONID}C
- Print the contents of the cookie JSESSIONID in the response sent to the client: %{JSESSIONID}o
- View and log the SSL cipher negotiated for each connection: "SSL=%{HTTPS}e" "%{HTTPS_CIPHER}e" "%{HTTPS_KEYSIZE}e" "%{HTTPS_SECRETKEYSIZE}e"
- Print the host name the request was for (useful when the site serves multiple hosts using virtual hosts): %{Host}i
**Access Log Response Times (%D)**

It is recommended to use %D in the LogFormat to track response times (in microseconds). The response time includes application time, queue time, and network time from/to the end-user and to/from the application.

Note that the time (%t) represents the time the request arrived for HTTPD >= 2.0 and the time the response was sent back for HTTPD < 2.0.

**Graphing Access Log Data**

To graph access log data, clone the problemdetermination git repository and run accesslog.sh (requires Perl and gnuplot) with the httpd.conf and access_log file:

```bash
$ git clone https://github.com/kgibm/problemdetermination
$ problemdetermination/scripts/ihs/accesslog.sh httpd.conf access_log
```

If you do not have the httpd.conf file, the script only needs two lines which you can put into an httpd.conf file (and customize LogFormat to match what httpd was run with):

```bash
$ echo -e "CustomLog common
LogFormat "%h %l %u %t \"%r\" %>s %b" common"
$ cat httpd.conf
CustomLog common
LogFormat "%h %l %u %t \"%r\" %>s %b" common
```

This will generate a PNG file analyzing response times, throughput, errors, and response bytes:
Access Log WAS Plugin Server Name (%{WAS}e)

If using the IBM WAS plugin, you can get the name of the application server that handled the request (http://publib.boulder.ibm.com/httpserv/ihsdiag/WebSphere61.html#LOG). The plugin sets an internal, per-request environment variable on the final transport it used to satisfy a request: %{WAS}e. It is fixed length so it has the first N characters of host/IP but always includes the port. The %{WAS}e syntax means log the environment variable (e) named 'WAS'.

The %{WAS}e LogFormat value may be plotted instead of WAS plugin STATS. Clone the problemdetermination git repository and run waspluginstatsfromaccess.sh (requires Perl and gnuplot) with the httpd.conf and access_log files:

```
$ git clone https://github.com/kgibm/problemdetermination
$ problemdetermination/scripts/ihs/waspluginstatsfromaccess.sh httpd.conf access_log
```
The web server plug-in contains a built-in ESI processor. The ESI processor can cache whole pages, as well as fragments, providing a higher cache hit ratio. The cache implemented by the ESI processor is an in-memory cache, not a disk cache, therefore, the cache entries are not saved when the web server is restarted.

When a request is received by the web server plug-in, it is sent to the ESI processor, unless the ESI processor is disabled. It is enabled by default. If a cache miss occurs, a Surrogate-Capabilities header is added to the request and the request is forwarded to the WebSphere Application Server. If servlet caching is enabled in the application server, and the response is edge cacheable, the application server returns a Surrogate-Control header in response to the WebSphere Application Server plug-in.

The value of the Surrogate-Control response header contains the list of rules that are used by the ESI processor to generate the cache ID. The response is then stored in the ESI cache, using the cache ID as the key. For each ESI "include" tag in the body of the response, a new request is processed so that each nested include results in either a cache hit or another request that forwards to the application server. When all nested includes have been processed, the page is assembled and returned to the client.

The ESI processor is configurable through the WebSphere web server plug-in configuration file plugin-cfg.xml. The following is an example of the beginning of this file, which
illustrates the ESI configuration options.

```
<Property Name="esiEnable" Value="true"/>
<Property Name="esiMaxCacheSize" Value="1024"/>
<Property Name="esiInvalidationMonitor" Value="false"/>
```

... The second option, esiMaxCacheSize, is the maximum size of the cache in 1K byte units. The default maximum size of the cache is 1 megabyte.

If the first response has a Content-Length response header, the web server plug-in checks for the response size. If the size of the response body is larger than the available ESI caching space, the response passes through without being handled by ESI.

Some parent responses have nested ESI includes. If a parent response is successfully stored in the ESI cache, and any subsequent nested include has a Content-length header that specifies a size larger than the available space in the ESI cache, but smaller than the value specified for esiMaxCacheSize property, the plug-in ESI processor evicts other cache elements until there is enough space for the nested include in the ESI cache.

The third option, esiInvalidationMonitor, specifies if the ESI processor should receive invalidations from the application server... There are three methods by which entries are removed from the ESI cache: first, an entry expiration timeout occurs; second, an entry is purged to make room for newer entries; or third, the application server sends an explicit invalidation for a group of entries. For the third mechanism to be enabled, the esiInvalidationMonitor property must be set to true and the DynaCacheEsi application must be installed on the application server. The DynaCacheEsi application is located in the installableApps directory and is named DynaCacheEsi.ear. If the ESIInvalidationMonitor property is set to true but the DynaCacheEsi application is not installed, then errors occur in the web server plug-in and the request fails.

This ESI processor is monitored through the CacheMonitor application. For the ESI processor cache to be visible in the CacheMonitor, the DynaCacheEsi application must be installed as described above, and the ESIInvalidationMonitor property must be set to true in the plugin-cfg.xml file.


If you're not using the ESI cache, disable it as it has some expensive operations in computing hashes for each request: Administrative Console -> Servers > Web Servers > web_server_name > Plug-in properties > Caching -> Uncheck "Enable ESI," and then re-generate and re-propagate plugin. ESI processing can also cause underisable buffering in the WAS Plug-in.

Elliptic Curve Cryptography (ECC) is available in TLS 1.2 and may be a faster algorithm than RSA for SSL signature and key exchange algorithms. As of 2012, ECC ciphers are not supported by most major web browsers, but they are supported by Java 7, OpenSSL, and GSKit. ECC ciphers start with TLS_EC and are available starting in IHS 8.0.0.6
KeepAlive

KeepAlive allows the client to keep a socket open between request, thus potentially avoiding TCP connection setup and tear down. For example, let's say a client opens a TCP connection and requests an HTML page. This HTML page contains one image. With KeepAlive, after the HTML response has been parsed and the image found, the client will re-use the previous TCP connection to request the image. ([http://publib.boulder.ibm.com/httpserv/manual70/mod/core.html#keepalive](http://publib.boulder.ibm.com/httpserv/manual70/mod/core.html#keepalive))

KeepAliveTimeout (default 5 seconds) is a balance between latency (a higher KeepAliveTimeout means a higher probability of connection re-use) and the maximum concurrently active requests (because a KeepAlive connection counts towards MaxClients for its lifetime).

ListenBacklog

The ListenBacklog is a suggestion to the OS TCP layer on the maximum number of connections to accept beyond MaxClients before refusing new connections (TCP SYN packets).

The maximum length of the queue of pending connections. Generally no tuning is needed or desired, however on some systems it is desirable to increase this when under a TCP SYN flood attack. See the backlog parameter to the listen(2) system call.

This will often be limited to a smaller number by the operating system. This varies from OS to OS. Also note that many OSES do not use exactly what is specified as the backlog, but use a number based on (but normally larger than) what is set.

[Gzip compression]

mod_deflate can be used to use gzip compression on responses:

mod_mpmstats

mpmstats ([http://publib.boulder.ibm.com/httpserv/ihsdiag/2.0/mod_mpmstats.html](http://publib.boulder.ibm.com/httpserv/ihsdiag/2.0/mod_mpmstats.html)) is a very simple but powerful httpd extension that periodically prints a line to error_log with a count of the number of threads that are ready, busy, keepalive, etc. Here's an example:

[Wed Jan 08 16:59:26 2014] [notice] mpmstats: rdy 48 bsy 3 rd 0 wr 3 ka 0
log 0 dns 0 cls 0

On z/OS, ensure PI24990 is installed.

The default mpmstats interval is 10 minutes although we recommend setting it to 30 seconds or less:

<IfModule mod_mpmstats.c>
# Write a record every 10 minutes (if server isn't idle).
# Recommendation: Lower this interval to 60 seconds, which will
# result in the error log growing faster but with more accurate
# information about server load.
ReportInterval 600
As covered in the mod_mpmstats link above, some of the key statistics are:

- rdy (ready): the number of web server threads started and ready to process new client connections
- bsy (busy): the number of web server threads already processing a client connection
- rd (reading): the number of busy web server threads currently reading the request from the client
- wr (writing): the number of busy web server threads that have read the request from the client but are either processing the request (e.g., waiting on a response from WebSphere Application Server) or are writing the response back to the client
- ka (keepalive): the number of busy web server threads that are not processing a request but instead are waiting to see if the client will send another request on the same connection; refer to the KeepAliveTimeout directive to decrease the amount of time that a web server thread remains in this state

**TrackHooks**

In recent versions, TrackHooks may be used to get per module response times, check for long-running modules, and track response times of different parts of the request cycle (http://publib.boulder.ibm.com/httpserv/ihsdiag/mpmstats_module_timing.html#loghooks):

Recommended mpmstats configuration

```plaintext
<IfModule mod_mpmstats.c>
  # Write a record to stderr every 10 seconds (if server isn't idle).
  ReportInterval 10
  TrackHooks allhooks
  TrackHooksOptions millis permodule logslow
  TrackModules On
  SlowThreshold 10
</IfModule>
```

Add the following to your LogFormat:

```plaintext
%{TRH}e %{TCA}e %{TCU}e %{TPR}e %{TAC}e %{RH}e
```

The final LogFormat line will most commonly look like this:

```plaintext
LogFormat "%h %l %u %t "%r" %>s %b %{TRH}e %{TCA}e %{TCU}e %{TPR}e %{TAC}e %{RH}e %{WAS}e %D" common
```

The above requires that mod_status and ExtendedStatus are enabled which enables additional statistics-gathering infrastructure in Apache:

```plaintext
LoadModule status_module modules/mod_status.so
<IfModule mod_status.c>
  ExtendedStatus On
</IfModule>
```

As long as the configuration does not use a "<Location /server-status> [...] SetHandler server-status [...] </Location>" block, then there is no additional security exposure by loading mod_status and enabling
ExtendedStatus (unless AllowOverride != ALL and someone creates a .htaccess file that enables it).

**Graphing mpmstats**

To graph mpmstats data, clone the problemdetermination git repository and run mpmstats.sh (requires Perl and gnuplot) with the error_log file:

```bash
$ git clone https://github.com/kgibm/problemdetermination
$ problemdetermination/scripts/ihs/mpmstats.sh error_log
```

This will generate a PNG file:

---

**mod_smf**

On z/OS, mod_smf provides additional SMF statistics:

**Status Module**

There is a [status module](http://publib.boulder.ibm.com/httpserv/manual70/mod/mod_smf.html) that can be enabled in IHS. It is not enabled by default (or it hasn't been in the past). However, it does present some interesting real time statistics which can help in understanding if requests are backing up or if the site is humming along nicely. It helps provide a second data point...
when trying to troubleshoot production problems. Most enterprise organizations will want to make sure the URL http://your.server.name/server-status?refresh=N to access the statistics are protected by a firewall and only available to the system administrators.


IHSDiag

Use ihsdiag to take thread dumps to understand what IHS threads are doing in detail:

Fast Response Cache Accelerator

FRCA/AFPA was deprecated starting in V7.0 [and] its use is discouraged. Instead, it is recommended to use the IBM HTTP Server default configuration to serve static files... If CPU usage with the default configuration is too high, the mod_mem_cache module can be configured to cache frequently accessed files in memory, or multiple web servers can be used to scale out horizontally. Additional options include the offloading of static files to a Content Delivery Network (CDN) or caching HTTP appliance, or to use the caching proxy component of WebSphere® Edge Server in WebSphere Application Server Network Deployment (ND). (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.ihs.doc/ihs/cihs_caoprestrict.html?cp=SSAW57_8.5.5&lang=en)

Websphere Plugin

ServerIOTimeout

Set a timeout value, in seconds, for sending requests to and reading responses from the application server.

If you set the ServerIOTimeout attribute to a positive value, this attempt to contact the server ends when the timeout occurs. However, the server is not [marked down].

If you set the ServerIOTimeout attribute to a negative value, the server is [marked down] whenever a timeout occurs...

If a value is not set for the ServerIOTimeout attribute, the plug-in, by default, uses blocked I/O to write requests to and read responses from the application server, and does not time out the TCP connection...

Setting the ServerIOTimeout attribute to a reasonable value enables the plug-in to timeout the connection sooner, and transfer requests to another application server when possible...

The default value is 900, which is equivalent to 15 minutes.
The ServerIOTimeout limits the amount of time the plug-in waits for each individual read or write operation to return. ServerIOTimeout does not represent a timeout for the overall request.

It is generally recommended to set a non-zero value for ServerIOTimeout. The value should be greater than the maximum expected response time for all legitimate requests.

In recent versions of WAS, the global ServerIOTimeout can be overridden for specific URLs (http://www-01.ibm.com/support/docview.wss?uid=swg1PM94198):

```
SetEnvIf Request_URI "\.jsp\$" websphere-serveriotimeout=10
```

By default, if a ServerIOTimeout pops, then the plugin will re-send non-affinity (http://www-01.ibm.com/support/docview.wss?uid=swg21450051) requests to the next available server in the cluster. If, for example, the request exercises a bug in the application that causes an OutOfMemoryError, then after each timeout, the request will be sent to all of the other servers in the cluster, and if the behavior is the same, then effectively it will lead to a complete, cascading failure. This behavior can be controlled with ServerIOTimeoutRetry:

```
ServerIOTimeoutRetry specifies a limit for the number of times the HTTP plugin retries an HTTP request that has timed out, due to ServerIOTimeout. The default value, -1, indicates that no additional limits apply to the number of retries. A 0 value indicates there are no retries. Retries are always limited by the number of available servers in the cluster.
Important: This directive does not apply to connection failures or timeouts due to the HTTP plugin ConnectTimeout.
```

The resolution of ServerIOTimeout may be affected by MaxSpareThreads. If ServerIOTimeout is taking longer than expected to fire, review the recommendations on MaxSpareThreads above and consider configuring it so that threads are not destroyed.

**Retries**

When will the WAS Plug-in retry a request:

http://publib.boulder.ibm.com/httpserv/ihsdiag/plugin_questions.html#retry

**Load Distribution**

Use LogLevel="Stats" to print load distribution in the plugin log after each request (see page 28):

http://www-01.ibm.com/support/docview.wss?uid=swg27020055&aid=1

To graph STATS data, clone the problemdetermination git repository and run waspluginstats.sh (requires Perl and gnuplot) with the http_plugin.log file:

```
$ git clone https://github.com/kgibm/problemdetermination
$ problemdetermination/scripts/ihs/waspluginstats.sh error_log
```
The WAS plugin log does not have a timezone in the timestamps, so you can set the time zone before running waspluginstats.sh:

```
$ export TZ=CDT
```

This will generate a PNG file:

![http_plugin.log totalRequests (Timezone CDT)](image)

There is an option called BackupServers which was used with WAS version 5 for DRS HTTP session failover, so this option is generally not used any more.

**MaxConnections**

You can limit the number of connections that can be handled by an applications server. To do this:

Go to the Servers > Server Types > WebSphere application servers > server_name.
In the Additional Properties section, click Web Server Plug-in properties.
Select Use maximum number of connections for the Maximum number of connections that can be handled by the Application server field.
Specify in the Connections field the maximum number of connections that you want to allow.
Then click Apply and Save.
When this maximum number of connections is reached, the plug-in, when establishing connections, automatically skips that application server, and tries the next available application server. If no application servers are available, an HTTP 503 response code will be returned to the client. This code indicates that the server is currently unable to handle the request because it is experiencing a temporary overloading or because maintenance is being performed.


In general, it is preferable to use the ODRlib plugin shipped in 8.5.5 instead of the classic WebSphere plugin because ODRlib has the Weighted Least Outstanding Request (WLOR) load balancing algorithm.

**WebSphere Caching Proxy (WCP)**

The WebSphere Caching Proxy (WCP) is optimized to store and serve cacheable responses from a backend application. WCP is primarily configured through the ibmproxy.conf file:


The CacheQueries directive may be specified multiple times with different patterns of URLs whose content may be cached. URL patterns may be excluded with the NoCaching directive.

```
CacheQueries PUBLIC http:///*/ *
NoCaching http://*/files/form/anonymous/api/library/*/document/*/media/*
```

HTTP responses may be GZIP compressed based on MIME type:

```
CompressionFilterEnable /opt/ibm/edge/cp/lib/mod_z.so
CompressionFilterAddContentType text/html
```

The CacheMemory directive specifies the maximum amount of native memory each WCP process may use for in-memory caching. This will be limited by the operating system, whether the process is 32-bit or 64-bit, shared libraries, and other constraints.

```
CacheMemory 1000 M
```

WCP has a thread pool which should match or exceed MaxClients in downstream web server(s) for example.

```
MaxActiveThreads 700
```

In general, it is recommended to pool the connections to the backend servers (such as web servers) to avoid the cost of constantly establishing and closing those connections.

```
ServerConnPool on
```

The time idle connections in this pool are held open is controlled with ServerConnTimeout and ServerConnGCRun.
By default, WCP will not cache responses with expiration times within the CacheTimeMargin. If you have available memory, disable this:

```
CacheTimeMargin 0
```

**Load Balancers**

Some load balancers are configured to keep affinity between the client IP address and particular web servers. This may be useful to simplify problem determination because the set of requests from a user will all be in one particular web server. However, IP addresses do not always uniquely identify a particular user (e.g. NAT), so this type of affinity can distort the distribution of requests coming into the web servers and it is not functionally required because the WAS plugin will independently decide how to route the request, including looking at request headers such as the JSESSIONID cookie if affinity is required to a particular application server.

Load balancers often have a probe function which will mark down back-end services if they are not responsive to periodic TCP or HTTP requests. One example of this happening was due to the load balancer performing TLS negotiation, exhausting its CPU, and then not having enough juice to process the response quickly enough.

**Applications**

**Java**

**Best Practices**

- Avoid the costs of object creation and manipulation by using primitive types for variables
- Cache frequently-used objects to reduce the amount of garbage collection needed, and avoid the need to re-create the objects.
- Group native operations to reduce the number of Java Native Interface (JNI) calls when possible.
- Use synchronized methods only when necessary to limit the multitasking in the JVM and operating system.
- Avoid invoking the garbage collector unless necessary. If you must invoke it, do so only during idle time or some noncritical phase.
- Declare methods as final whenever possible. Final methods are handled better by the JVM.
- Use the static final key word when creating constants in order to reduce the number of times the variables need to be initialized.
- Avoid unnecessary "casts" and "instanceof" references, because casting in Java is done at run time.
- Avoid the use of vectors whenever possible when an array will suffice.
- Add and delete items from the end of the vector.
- Avoid allocating objects within loops.
- Use connection pools and cached-prepared statements for database access.
- Minimize thread creation and destruction cycles.
• Minimize the contention for shared resources.
• Minimize the creation of short-lived objects.
• Avoid remote method calls.
• Use callbacks to avoid blocking remote method calls.
• Avoid creating an object only used for accessing a method.
• Keep synchronized methods out of loops.
• Store string and char data as Unicode in the database.
• Reorder the CLASSPATH so that the most frequently used libraries occur first.
• Reduce synchronization
• Keep application logging to a minimum or add log guards
• Consider using work areas for passing around application state through JNDI:

Synchronization

"Problem determination... tools often report the class of the object on which contention is occurring. A uniquely named class for the object helps identify where in the application code those objects are being used." (http://www.ibm.com/developerworks/websphere/techjournal/1111_dawson/1111_dawson.html)

Applications that overuse the synchronized keyword or have one placed in a frequently used method can often result in poor application response times and/or application deadlocks. Applications should be written to be thread safe (http://www.ibm.com/developerworks/java/library/j-jtp09263/index.html).

java.lang.ThreadLocal

ThreadLocals are a powerful way to cache information without incurring cross thread contention and also ensuring thread safety of cached items. When using ThreadLocals in thread pools, consider ensuring that the thread pool minimum size is equal to the thread pool maximum size, so that ThreadLocals are not destroyed. (http://docs.oracle.com/javase/7/docs/api/java/lang/ThreadLocal.html)

Note that ThreadLocals may introduce classloader leaks if the ThreadLocal object (or an object it references) is loaded from an application classloader which is restarted without the JVM being restarted. In this case, the only way to clear ThreadLocals is to allow those threads to be destroyed or the ThreadLocal values to be updated to a class from the new classloader (this can be done with a module listener).

Speculative Tracing

ThreadLocals may be used to speculatively dump trace statements. For example, the following code only dumps trace statements if the total duration of the method call exceeds some threshold. This code is most efficient when run within fixed-sized thread pools.

```java
private static final int SPECULATIVE_DURATION_THRESHOLD =
    Integer.parseInt(System.getProperty("SPECULATIVE_DURATION_THRESHOLD", "-1"));
private static final boolean SPECULATIVE_TRACE_ENABLED =
    SPECULATIVE_DURATION_THRESHOLD == -1 ? false : true;
private static ThreadLocal<ArrayList<String>> speculativeTraces = new ThreadLocal<ArrayList<String>>() {
    @Override
    public ArrayList<String> get()
```
protected ArrayList<String> initialValue() {
    return new ArrayList<String>(8);
}

public void foo() {
    final long methodStartTime = SPECULATIVE_TRACE_ENABLED ?
        System.currentTimeMillis() : -1;
    final ArrayList<String> spec = SPECULATIVE_TRACE_ENABLED ?
        speculativeTraces.get() : null;
    if (SPECULATIVE_TRACE_ENABLED) {
        spec.clear();
        spec.add(methodStartTime + " started");
    }
    doWork1();
    if (SPECULATIVE_TRACE_ENABLED) {
        spec.add(System.currentTimeMillis() + " doWork1 finished");
    }
    doWork2();
    if (SPECULATIVE_TRACE_ENABLED) {
        spec.add(System.currentTimeMillis() + " doWork2 finished");
    }
    doWork3();
    if (SPECULATIVE_TRACE_ENABLED) {
        final long methodDuration = System.currentTimeMillis() - methodStartTime;
        if (methodDuration >= SPECULATIVE_DURATION_THRESHOLD) {
            System.out.println("Speculative tracing threshold (" +
                SPECULATIVE_DURATION_THRESHOLD + " ms) exceeded with a call of " +
                methodDuration + " ms");
            for (String speculativeTrace : spec) {
                System.out.println(speculativeTrace);
            }
            System.out.println("Speculative tracing set end at " +
                System.currentTimeMillis());
        }
    }
}

Sampled Timing Calls
ThreadLocals may be used to sample long method call execution times and print out sampled
durations:

    private static final int SAMPLE_COUNTPERTHREAD_FOO =
        Integer.getInteger("samplecountperthread.foo", 1000);
    private static final int THRESHOLD_FOO =
        Integer.getInteger("threshold.foo", 0);
    private static final String SAMPLE_MESSAGE_FOO = "Sampled duration (threshold=" +
        THRESHOLD_FOO + ", rate=")
private static final AtomicInteger calls_foo = new AtomicInteger(0);

public void foo() {
    final boolean doSample = (calls_foo.incrementAndGet() % SAMPLE_COUNTPERTHREAD_FOO) == 0;
    final long startTime = doSample ? System.currentTimeMillis() : -1;
    doLongWork();
    if (doSample) {
        final long diff = System.currentTimeMillis() - startTime;
        if (diff >= THRESHOLD_FOO) {
            System.out.println(SAMPLE_MESSAGE_FOO + diff);
        }
    }
}

private void doLongWork() {
    // ...
}

### Always Timing Calls

As an alternative to sampling, you may add code to time all calls and then print details if they are above some threshold:

```java
private static final int THRESHOLD_FOO = Integer.getInteger("threshold.foo", 0);
private static final String TIMING_MESSAGE_FOO = "Duration (threshold=" + THRESHOLD_FOO + ") in ms of foo = ";

public void foo() {
    final long startTime = System.currentTimeMillis();
    doLongWork();
    final long diff = System.currentTimeMillis() - startTime;
    if (diff >= THRESHOLD_FOO) {
        System.out.println(TIMING_MESSAGE_FOO + diff);
    }
}

private void doLongWork() {
    // ...
}
```

### Requesting Thread Dumps, Heap Dumps, and System Dumps

The following example code shows how to request a thread dump (IBM Java only), heap dump or system dump:
/**
 * These are handled in synchronized methods below.
 */
private static int threadDumpsTaken = 0, heapDumpsTaken = 0, coreDumpsTaken = 0;

private static final boolean isIBMJava;
private static final Class<?> ibmDumpClass;
private static final java.lang.reflect.Method ibmJavacoreMethod;
private static final java.lang.reflect.Method ibmHeapDumpMethod;
private static final java.lang.reflect.Method ibmSystemDumpMethod;
private static final Class<?> hotSpotMXBeanClass;
private static final Object hotspotMXBean;
private static final java.lang.reflect.Method hotspotMXBeanDumpHeap;
private static final java.text.SimpleDateFormat hotspotDateFormat = new java.text.SimpleDateFormat("yyyyMMdd'T'HHmmss");

static {
    try {
        isIBMJava = isIBMJava();
        ibmDumpClass = isIBMJava ? Class.forName("com.ibm.jvm.Dump") : null;
        ibmHeapDumpMethod = isIBMJava ? ibmDumpClass.getMethod("HeapDump") : null;
        ibmJavacoreMethod = isIBMJava ? ibmDumpClass.getMethod("JavaDump") : null;
        ibmSystemDumpMethod = isIBMJava ? ibmDumpClass.getMethod("SystemDump") : null;
        hotSpotMXBeanClass = isIBMJava ? null : getHotSpotDiagnosticMXBeanClass();
        hotspotMXBean = isIBMJava ? null : getHotSpotDiagnosticMXBean();
        hotspotMXBeanDumpHeap = isIBMJava ? null : getHotSpotDiagnosticMXBeanDumpHeap();
    } catch (Throwable t) {
        throw new RuntimeException("Could not load Java dump classes", t);
    }
}

public static boolean isIBMJava() {
    try {
        // We could use System.getProperty, but that requires elevated permissions in some cases.
        Class.forName("com.ibm.jvm.Dump");
        return true;
    } catch (Throwable t) {
        return false;
    }
}

private static Class<?> getHotSpotDiagnosticMXBeanClass() throws
ClassNotFoundException {
    return Class.forName("com.sun.management.HotSpotDiagnosticMXBean");
}

private static Object getHotSpotDiagnosticMXBean() throws ClassNotFoundException, java.io.IOException {
    javax.management.MBeanServer server =
    java.lang.management.ManagementFactory.getPlatformMBeanServer();
    return java.lang.management.ManagementFactory.newPlatformMXBeanProxy(server,
            "com.sun.management:type=HotSpotDiagnostic", hotSpotMXBeanClass);
}

private static java.lang.reflect.Method getHotSpotDiagnosticMXBeanDumpHeap() throws NoSuchMethodException, SecurityException {
    return hotSpotMXBeanClass.getMethod("dumpHeap", String.class, boolean.class);
}

public static synchronized void requestThreadDump() {
    if (maxThreadDumps == -1 || (maxThreadDumps > -1 && threadDumpsTaken++ < maxThreadDumps)) {
        try {
            ibmJavacoreMethod.invoke(ibmDumpClass);
        } catch (Throwable t) {
            throw new RuntimeException(t);
        }
    }
}

public static synchronized void requestHeapDump() {
    if (maxHeapDumps == -1 || (maxHeapDumps > -1 && heapDumpsTaken++ < maxHeapDumps)) {
        try {
            if (ibmHeapDumpMethod != null) {
                ibmHeapDumpMethod.invoke(ibmDumpClass);
            } else {
                requestHotSpotHPROF();
            }
        } catch (Throwable t) {
            throw new RuntimeException(t);
        }
    }
}

public static synchronized void requestCoreDump() {
    if (maxCoreDumps == -1 || (maxCoreDumps > -1 && coreDumpsTaken++ < maxCoreDumps)) {
        try {
            if (ibmSystemDumpMethod != null) {
                ibmSystemDumpMethod.invoke(ibmDumpClass);
            } else {
                requestHotSpotHPROF();
            }
        } catch (Throwable t) {
            throw new RuntimeException(t);
        }
    }
}
private static void requestHotSpotHPROF() throws IllegalAccessException,
java.lang.reflect.InvocationTargetException {
    String fileName = "heap" + hotspotDateFormat.format(new java.util.Date()) + ".hprof";
    boolean live = true;
    hotspotMXBeanDumpHeap.invoke(hotspotMXBean, fileName, live);
}

does not

java.util.logging

Example of how to use java.util.logging:

package com.test;

public class Foo {
    private static final java.util.logging.Logger LOG =
    java.util.logging.Logger.getLogger(Foo.class.getName());

    public void bar(String param1) {
        if (LOG.isLoggable(java.util.logging.Level.FINE)) {
            LOG.entering(Foo.class.getName(), "bar", param1);
        }

        // Do work...

        if (LOG.isLoggable(java.util.logging.Level.FINER)) {
            LOG.finer("Work step1 complete");
        }

        // Do work...

        if (LOG.isLoggable(java.util.logging.Level.FINE)) {
            LOG.entering(Foo.class.getName(), "bar");
        }
    }
}

For example, the logging or trace specification may control the logging of this class with
com.test.Foo=all

Finalizers

"The Java service team recommends that applications avoid the use of finalizers if possible."
(http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.win.80.doc/diag/understa
nding/mm_gc_finalizers.html)

"It is not possible to predict when a finalizer is run... Because a finalized object might be garbage that
is retained, a finalizer might not run at all."
(http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.win.80.doc/diag/understa
XML Parsers

One of the common misconceptions about writing XML applications is that creating a parser instance does not incur a large performance cost. On the contrary, creation of a parser instance involves creation, initialization, and setup of many objects that the parser needs and reuses for each subsequent XML document parsing. These initialization and setup operations are expensive.

In addition, creating a parser can be even more expensive if you are using the JAXP API. To obtain a parser with this API, you first need to retrieve a corresponding parser factory -- such as a SAXParserFactory -- and use it to create the parser. To retrieve a parser factory, JAXP uses a search mechanism that first looks up a ClassLoader (depending on the environment, this can be an expensive operation), and then attempts to locate a parser factory implementation that can be specified in the JAXP system property, the jaxp.property file, or by using the Jar Service Provider mechanism. The lookup using the Jar Service Provider mechanism can be particularly expensive as it may search through all the JARs on the classpath; this can perform even worse if the ClassLoader consulted does a search on the network.

Consequently, in order to achieve better performance, we strongly recommend that your application creates a parser once and then reuses this parser instance.


Apache HttpClient

The process of establishing a connection from one host to another is quite complex and involves multiple packet exchanges between two endpoints, which can be quite time consuming. The overhead of connection handshaking can be significant, especially for small HTTP messages [and particularly for TLS]. One can achieve a much higher data throughput if open connections can be re-used to execute multiple requests... HttpClient fully supports connection persistence.


Web Applications

It is important to reduce the number of resources (images, CSS, Javascript, etc.) served for each request (caching and compression are also important, dealt elsewhere in the Cookbook). You can use browser or network sniffing tools to determine the largest number and sizes of resources. Here are some examples:

1. Consider combining images into a single image - often called a "sprite" - and display those images using CSS sprite offset techniques.
2. Consider combining multiple JavaScript files into a single file.
3. Consider "minifying" JavaScript and CSS files.
4. Consider compressing or resizing images more.

**HTTP Sessions**

Individual sessions retaining more than 1MB may be concerning. Use a system dump or heap dump and a tool such as the Memory Analyzer Tool with the IBM Extensions for Memory Analyzer to deep dive into session sizes and contents (http://www.ibm.com/developerworks/websphere/techjournal/0405_brown/0405_brown.html).

If there is a logout link, call javax.servlet.http.HttpSession.invalidate() to release the HTTP session as early as possible, reducing memory pressure:

If using session persistence, consider implementing manual update and sync of session updates:

Keep the amount of data in the HTTP session as small as possible.

Only touch session attributes that actually change. This allows for administrative changes to only persist updated attributes to the HTTP Session persistent storage.

**Tuning the Application from the Client Side**

The IBM Page Detailer tool helps analyze client side performance:
https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=61d74777-1701-4014-bfc0-96067ed50156. One of the benefits for doing this is to help identify static objects that are not being cached at the browser (i.e. you see frequent fetches of the same .gif or .jpg file). In addition, for JSF based applications, this tool helps identify large client side caches which can cause longer response times.

**Database Access**

SQL statements should be written to use the parameterized ? (question mark) notation. In order for the prepared statement cache to be used effectively the parameterized statements will be reused from the cache. Consequently, building SQL statements with the parameters substituted in will all look like different statements and the cache will have little performance effect.

If you are using global transactions, use deferred enlistment:

Make sure to close Connections, Statements, and ResultSets. In some databases (e.g. https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/rprf_wastundb2.html), not closing all of these may cause additional overhead even if the objects will ultimately be closed by the pools.
JDBC Deadlocks

Applications that open more than one JDBC connection to the same datasource can result in a deadlock if there are not enough connections in the connection pool. See [http://www-01.ibm.com/support/docview.wss?uid=swg1JR43775](http://www-01.ibm.com/support/docview.wss?uid=swg1JR43775) If javacores show multiple threads waiting for a connection and WebSphere Application Server is reporting hung threads then you will want to increase the number of connections in the connection pool to at least 2n+1 where n = maximum number of threads in the thread pool. Applications that open more than 2 connections to the same datasource will need even larger pools (3n+1, 4n+1, etc).

To correct this problem the application developer has to fix the code to close a JDBC connection before opening another JDBC connection.

Web Services

Provide a jaxb.index file for every package that does not contain an ObjectFactory class. This action enables the system to completely avoid the search for JAXB classes. This approach does require application modification to account for the addition of the jaxb.index files. ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cwbs_tuning_jaxbcontext.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/cwbs_tuning_jaxbcontext.html))

Service Component Architecture (SCA)

Use @AllowsPassByReference if possible with SCA modules: [https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tsca_passby_ref.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tsca_passby_ref.html)

Object Caching

The DistributedMap and DistributedObjectCache interfaces are simple interfaces for the dynamic cache. Using these interfaces, Java EE applications and system components can cache and share Java objects by storing a reference to the object in the cache. The default dynamic cache instance is created if the dynamic cache service is enabled in the administrative console. This default instance is bound to the global Java Naming and Directory Interface (JNDI) namespace using the name services/cache/distributedmap. ([https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tdyn_distributedmap.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tdyn_distributedmap.html))

Tread with caution. Overly active distributed maps can become quite chatty amongst the JVMs and, in extreme cases, limit the total number of JVMs in the same distributed map domain because the JVMs spend most of their time chatting about the changes that occurred in the map.

WeakReferences and SoftReferences

A typical use of the SoftReference class is for a memory-sensitive cache. The idea of a SoftReference is that you hold a reference to an object with the guarantee that all of your soft references will be cleared before the JVM reports an out-of-memory condition. The key point is that when the garbage collector runs, it may or may not free an object that is
softly reachable. Whether the object is freed depends on the algorithm of the garbage collector as well as the amount of memory available while the collector is running. The WeakReference class

A typical use of the WeakReference class is for canonicalized mappings. In addition, weak references are useful for objects that would otherwise live for a long time and are also inexpensive to re-create. The key point is that when the garbage collector runs, if it encounters a weakly reachable object, it will free the object the WeakReference refers to. Note, however, that it may take multiple runs of the garbage collector before it finds and frees a weakly reachable object.


Logging

Always use a logger that can be dynamically modified at run time without having to restart the JVM. Differentiate between Error logging (which should go to SystemOut.log) and Audit logging which has different requirements and should not be contaminating the SystemOut.log. Use a FAST disk for Audit logging.

Application Development

Rational Application Developer

Tuning the workspace


The configuration changes listed below can be done to improve RAD’s performance depending on individual needs. Some of these tips have been integrated to the product into the Workspace performance tuning feature, available by clicking Help > Performance > Workspace performance tuning.

• If you already have some projects in the workspace, published or not into WebSphere Application Server v7.0 to 8.5.5, you could start by using the Workspace performance tuning tool.
• The table in the Performance Tips at the Knowledge Center summarizes the tips, points out the type of improvement and how frequently any of these configuration changes are considered. The table below suggests which configurations might be useful for a few typical conditions you could have in your workspace. This does not mean the tip is exclusive for that condition though.
• Most of these tips can be used also in WDT. Look below the table for a brief description of each task. A special note is done on those that are only for RAD.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many projects and/or files in the workspace</td>
<td>• Convert projects to binary form</td>
</tr>
<tr>
<td>Condition</td>
<td>Tip</td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
</tr>
<tr>
<td>• Closing Projects&lt;br&gt;• Validation&lt;br&gt;• Automatically build and refresh the workspace&lt;br&gt;• Links&lt;br&gt;• Plug-ins activated on startup</td>
<td></td>
</tr>
<tr>
<td><strong>Workspace is old / has had many changes</strong></td>
<td>• Fresh workspaces</td>
</tr>
<tr>
<td>&lt;br&gt;<strong>Limited resources</strong>&lt;br&gt;• Do not install features that are not required&lt;br&gt;• Remote test server&lt;br&gt;• Restarting Rational Application Developer&lt;br&gt;• JVM tuning&lt;br&gt;• JVM tuning - shared classes&lt;br&gt;• Capabilities&lt;br&gt;• Reducing memory&lt;br&gt;• Quick Diff&lt;br&gt;• Label decorations</td>
<td></td>
</tr>
<tr>
<td><strong>Constantly modifying projects published to WAS 7.0 to 8.5.5 traditional profile</strong></td>
<td>&lt;br&gt;• Publishing and annotations&lt;br&gt;• Server configuration options&lt;br&gt;• Server Startup Options (Admin Console)&lt;br&gt;• Restarting projects</td>
</tr>
<tr>
<td><strong>General tips</strong></td>
<td>&lt;br&gt;• Defragmenting&lt;br&gt;• Antivirus software&lt;br&gt;• Task Manager</td>
</tr>
</tbody>
</table>

**Automatically build and refresh the workspace**

When "Build Automatically" is enabled, each time files are saved, a build is triggered, which makes the save operation itself take longer. "Refresh Automatically" is only useful if constantly working with external editors to modify files in the workspace. If not, this will only spend resources in monitoring for changes caused by external processes.

RAD: Make sure that "Build Automatically" and "Refresh Automatically" options are disabled in **Window > Preferences > General > Workspace**.

WDT: "Build Automatically" exists in the same wizard, but there are two options instead of "Refresh Automatically": "Refresh using native hooks or polling" and "Refresh on access" ([http://help.eclipse.org/kepler/topic/org.eclipse.platform.doc.user/tasks/tasks-52.htm?cp=0_3_3_8](http://help.eclipse.org/kepler/topic/org.eclipse.platform.doc.user/tasks/tasks-52.htm?cp=0_3_3_8)). "Refresh on access" gets activated only when a file is opened. Make sure to have at least "Build
Automatically" and "Refresh using native hooks or polling" disabled.

**Convert projects to binary form**
If a workspace is large and has a lot of projects that are not frequently updated, convert those to binary form. This will reduce memory footprint and speed up development tasks:

**Capabilities**
By disabling capabilities, you can prevent invoking an unneeded function and save time and memory resources by not having a plugin you don't need loaded. You can enable/disable capabilities at **Window > Preferences > General > Capabilities.**

**Closing Projects**
Any projects in the workspace that are not being modified or needed as dependencies of other projects should be deleted or closed. While they remain open, time and memory are consumed in constantly building and validating their source code.

**To close it:** right click the project and select **Close Project.**

**Defragmenting**
Defragmenting helps with the startup of the product, and also with some I/O intensive, like build and validation. Only available in Windows.

**Do not install features that are not required**
This will reduce memory footprint and also save time that could be consumed during activation of plugins.

**Plug-ins activated on startup**
There are plug-ins in RAD that need to always be activated on startup in order to enable some functions. One of these plug-ins is the **IBM Common Migration UI**, which, when migrating resources into the workspace, detects and suggest changes to those projects if needed. If you have already performed the migration and are working with a large workspace, you can opt to disable the **IBM Common Migration UI** by clearing its option in **Window > Preferences > General > Startup and Shutdown.**

**Fresh workspaces**
In some cases, where a workspace is old, the workspace metadata can accumulate and impact performance. Creating a new workspace can help with this, but is important to note that if you've set preferences in your workspace, you will need to set them again on the new workspace. You may also export the preferences and import them into the new workspace.

**JVM tuning**
The location of the JVM tuning parameters is the **eclipse.ini** file in the installation directory.
RAD Comes tuned for what's been considered the average workspace.

**JVM tuning - shared classes**
Can improve product's startup time. Note: Only applies to IBM JVM. RAD on the MAC ships the Oracle JVM.

**Label decorations**
Label Decorations allow additional information to be displayed in an item's label and icon. Disabling all or some decorations can have a little improvement in performance.
http://help.eclipse.org/kepler/topic/org.eclipse.platform.doc.user/reference/ref-decorations.htm?cp=0_4_1_33

**Links (The Link Indexer)**
The Link Indexer monitors hyperlinks. It can be disabled by clearing **Supply link results to Java search** in **Window > Preferences > Web > Links.** Or you can just exclude some resources from indexing. Some activities like link refactoring depend on this function to work appropriately. As a
possible rule of thumb: if there's a lot of hyperlinks in the workspace and you won't be refactoring, you
can disable this.

**Publishing and annotations**
RAD: This task can also be done automatically using the **Workspace performance tuning** tool.
For Web 2.5 applications that do not contain annotations, you can reduce the time to publish by setting
the **metadata-complete** property on the **WebContent/WEB-INF/web.xml** file to true.
If a project contains annotations, you can use the directives
`com.ibm.ws.amm.scan.context.filter.archives` and `com.ibm.ws.amm.scan.context.filter.packages` to
prevent the server to scan certain JAR files or packages
doc/ae/xrun_jvm.html).

**Quick Diff**
You can get a small performance improvement by disabling Quick Diff: **Window > Preferences >**
**General > Editors > Text Editors > Quick Diff**. Quick Diff displays a marker on the editor indicating
changes done since last file save.

**Remote test server**
You can run the test server on the second system to free up resources on your development machine.
l for how to create a server and
http://www.ibm.com/support/knowledgecenter/SSHR6W_8.5.5/com.ibm.websphere.wdt.doc/topics/termote_start.htm for how to enable a server to be started remotely.

**Restarting projects**
l

**Server configuration options**
html

**Server Startup Options (Admin Console)**
To improve server startup performance, ensure that the Run in development mode and Parallel start are
selected. Also remove applications that are not required from the installed applications list.

**Reducing memory**
RAD only: Click **Help > Performance >** **Reduce Memory**.

**Restarting Rational Application Developer**
As with other applications, some memory can be freed up by restarting it, but some considerations
should be taken if the workspace is really large:
- Consider disabling automatic builds.
- Suspend all validators.
- Consider disabling link indexer.

**Validation**
You can reduce the build time by disabling some or all validators at **Window > Preferences >**

Workspace performance tuning

The Workspace performance tuning tool implements a series of tasks that examine the workspace and make recommendations for changes (or in some cases, do the actual changes) to have a better performance.


Containers

See sub-chapters:

- Docker
- Kubernetes
- IBM Cloud

Tutorials: [https://github.com/WASdev/ci.docker.tutorials](https://github.com/WASdev/ci.docker.tutorials)

Docker

WebSphere Liberty images: [https://hub.docker.com/ /websphere-liberty](https://hub.docker.com/ /websphere-liberty)

Open Liberty images: [https://hub.docker.com/ /open-liberty](https://hub.docker.com/ /open-liberty)

Traditional WebSphere images: [https://hub.docker.com/r/ibmcom/websphere-traditional/](https://hub.docker.com/r/ibmcom/websphere-traditional/)

Splash-Only WebSphere Liberty

Running a simple WebSphere Liberty container:

```
docker run -p 80:9080 -p 443:9443 websphere-liberty:webProfile8
```

Startup is complete when you see:

```
[AUDIT ] CWWKF0011I: The server defaultServer is ready to run a smarter planet.
```

Access the default splash page at [http://localhost/](http://localhost/) or [https://localhost/](https://localhost/)

Simple Web Application File

Extend the above example by mounting a .war or .ear file from the host to /config/dropins/. For example, using [https://raw.githubusercontent.com/kgibm/java_web_hello_world/master/builds/java_web_hello_world.ear](https://raw.githubusercontent.com/kgibm/java_web_hello_world/master/builds/java_web_hello_world.ear)

```
docker run -p 80:9080 -p 443:9443 -v ~/Downloads/java_web_hello_world.ear:/config/dropins/java_web_hello_world.ear
```

Page 467
The default context root is the basename of the .war or .ear file unless a specific context root has been configured. In this example, the context root is / so access this example at http://localhost/.

The host path must be an absolute path, so if it's in the current or relative directory, use $(pwd):

```bash
docker run -p 80:9080 -p 443:9443 -v $(pwd)/java_web_hello_world.ear:/config/dropins/java_web_hello_world.ear
websphere-liberty:webProfile8
```

Source code of the example application: https://github.com/kgibm/java_hello_world/

### Traditional WAS

Create a file in the current directory named PASSWORD with an administrative password as its contents. For example:

```bash
wsadmin
```

Then run:

```bash
docker run -p 9043:9043 -p 9443:9443 -v $(pwd)/PASSWORD:/tmp/PASSWORD -e ENABLE_BASIC_LOGGING=true ibmcom/websphere-traditional:latest
```

After you see "open for e-business", access the administrative console with the user name wsadmin and the password from the PASSWORD file at https://localhost:9043/ibm/console/login.do?action=secure

Access the WebContainer port at 9443: https://localhost:9443/snoop

### Managing Containers

Use the `-d` flag to start the container in the background.

```bash
docker run -d -p 80:9080 -p 443:9443 websphere-liberty:webProfile8
```

### Commands

Print the container IDs of all running containers that are based on the websphere-liberty:webProfile8 image:

```bash
docker ps -f "ancestor=websphere-liberty:webProfile8" --format "{{.ID}}"
```

The output of the above command may be sub-shelled into other docker commands:

"Log in" to the container:

```bash
docker exec -it $(docker ps -f "ancestor=websphere-liberty:webProfile8" --format "{{.ID}}") bash
```

Print messages.log:

```bash
docker exec -it $(docker ps -f "ancestor=websphere-liberty:webProfile8" --format "{{.ID}}") cat /logs/messages.log
```
Tail messages.log:

docker exec -it $(docker ps -f "ancestor=websphere-liberty:webProfile8" --format "{.ID}") tail -f /logs/messages.log

Adding server.xml Configuration

The base server.xml configuration may be specified with .xml files in /config/configDropins/defaults/
Overriding server.xml configuration may be specific with .xml files in /config/configDropins/overrides/

HTTP Access Log Configuration

accesslog.xml

<server>
    <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080" httpsPort="9443">  
      <accessLogging filepath="${server.output.dir}/logs/access.log" maxFileSize="250" maxFiles="4" logFormat="%h %i %u %t "%r" %s %b %D" />
    </httpEndpoint>
</server>

Running example:

docker run -p 80:9080 -p 443:9443 -v $(pwd)/accesslog.xml:/config/configDropins/overrides/accesslog.xml websphere-liberty:webProfile8

Example request:

$ curl -I http://localhost/  
HTTP/1.1 200 OK  
Cache-Control: max-age=604800  
Connection: Close  
Content-Type: text/html; charset=UTF-8  
Date: Mon, 28 Jan 2019 22:01:24 GMT

Printing the access log:

$ docker exec -it $(docker ps -f "ancestor=websphere-liberty:webProfile8" --format "{.ID}") cat /opt/ibm/wlp/output/defaultServer/logs/access.log  

Request Timing Configuration

requestTiming.xml: (Note that 0ms or 1ms don't always work):

<server>
    <featureManager><feature>requestTiming-1.0</feature></featureManager>
    <requestTiming slowRequestThreshold="2ms" hungRequestThreshold="60s" sampleRate="1" />
</server>
Running example:

docker run -p 80:9080 -p 443:9443 -v $(pwd)/swat.ear:/config/dropins/swat.ear
-v $(pwd)/requestTiming.xml:/config/configDropins/overrides/requestTiming.xml
websphere-liberty:webProfile8

Example request:

$ ( time curl -s -o /dev/null http://localhost/swat/Sleep?duration=1000 )
2>&1 | grep real
real 0m1.028s

Output in the console or messages.log:

$ docker exec -it $(docker ps -f "ancestor=websphere-liberty:webProfile8"
--format "{{.ID}}") tail -f /logs/messages.log
[1/29/19 17:30:52.953 UTC] 0000004c
com.ibm.ws.request.timing.manager.SlowRequestManager W TRAS0112W:
Request AAABDRku0li_AAAAAAA has been running on thread 00000045
for at least 24.520ms. The following stack trace shows what this thread is
currently running.
    at
com.ibm.ws.webcontainer.srt.SRTServletRequest.getDispatchContext(SRTServletRequest.java:4042)
    at
com.ibm.ws.webcontainer.srt.SRTServletRequest.getUserPrincipal(SRTServletRequest.java:2827)
    at
com.ibm.ws.webcontainer.srt.SRTServletRequest.getRemoteUser(SRTServletRequest.java:2106)
    at
com.ibm.BaseServlet.getUser(BaseServlet.java:131)
    at
com.ibm.BaseServlet.service(BaseServlet.java:64)
    at
javax.servlet.http.HttpServlet.service(HttpServlet.java:791)
    at
com.ibm.ws.webcontainer.servlet.ServletWrapper.service(ServletWrapper.java:1255)
    at
com.ibm.ws.webcontainer.servlet.ServletWrapper.handleRequest(ServletWrapper.java:743)
    at
com.ibm.ws.webcontainer.servlet.ServletWrapper.handleRequest(ServletWrapper.java:440)
    at
com.ibm.ws.webcontainer.filter.WebAppFilterChain.invokeTarget(WebAppFilterChain.java:182)
    at
    at
    at
com.ibm.ws.webcontainer.filter.FilterInstanceWrapper.doFilter(FilterInstanceWrapper.java:201)
    at
com.ibm.ws.webcontainer.filter.WebAppFilterChain.doFilter(WebAppFilterChain.j
ava:90)
at com.ibm.ws.webcontainer.filter.WebAppFilterManager.doFilter(WebAppFilterManager.java:996)
at com.ibm.ws.webcontainer.filter.WebAppFilterManager.invokeFilters(WebAppFilterManager.java:1134)
at com.ibm.ws.webcontainer.webapp.WebApp.handleRequest(WebApp.java:4954)
at com.ibm.ws.webcontainer.osgi.DynamicVirtualHost$2.handleRequest(DynamicVirtualHost.java:314)
at com.ibm.ws.webcontainer.osgi.DynamicVirtualHost$2.run(DynamicVirtualHost.java:279)
at com.ibm.ws.http.dispatcher.internal.channel.HttpDispatcherLink.wrapHandlerAndExecute(HttpDispatcherLink.java:417)
at com.ibm.ws.http.channel.internal.inbound.HttpInboundLink.handleDiscrimination(HttpInboundLink.java:532)
at com.ibm.ws.http.channel.internal.inbound.HttpInboundLink.handleNewRequest(HttpInboundLink.java:466)
at com.ibm.ws.http.channel.internal.inbound.HttpInboundLink.processRequest(HttpInboundLink.java:331)
at com.ibm.ws.http.channel.internal.inbound.HttpInboundLink.ready(HttpInboundLink.java:302)
at com.ibm.ws.tcpchannel.internal.NewConnectionInitialReadCallback.sendToDiscriminators(NewConnectionInitialReadCallback.java:165)
at com.ibm.ws.tcpchannel.internal.NewConnectionInitialReadCallback.complete(NewConnectionInitialReadCallback.java:74)
at com.ibm.ws.tcpchannel.internal.WorkQueueManager.requestComplete(WorkQueueManager.java:501)
at com.ibm.ws.tcpchannel.internal.WorkQueueManager.attemptIO(WorkQueueManager.java:571)
at com.ibm.ws.tcpchannel.internal.WorkQueueManager.workerRun(WorkQueueManager.java:926)
at com.ibm.ws.tcpchannel.internal.WorkQueueManager$Worker.run(WorkQueueManager.java:1015)
at
The following table shows the events that have run during this request.

<table>
<thead>
<tr>
<th>Duration</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>29.885ms +</td>
<td>websphere.servlet.service</td>
</tr>
</tbody>
</table>

**Event Logging Configuration**

The eventLogging-1.0 feature is not bundled with the websphere-liberty images so you must build an image on top:

```
eventLogging.xml
<server>
  <featureManager><feature>eventLogging-1.0</feature></featureManager>
  <eventLogging includeTypes="websphere.servlet.service" minDuration="0"
    logMode="exit" sampleRate="1" />
</server>
```

Running example:

```
printf "FROM websphere-liberty:webProfile8\nRUN
/opt/ibm/wlp/bin/installUtility install --acceptLicense eventLogging-1.0\n" |
docker build -t websphere-liberty-plus -f - .
docker run -p 80:9080 -p 443:9443 -v $(pwd)/eventLogging.xml:/config/configDropins/overrides/eventLogging.xml -v $(pwd)/swat.ear:/config/dropins/swat.ear websphere-liberty-plus
```

Example request:

```
$ ( time curl -s -o /dev/null http://localhost/swat/Sleep?duration=1000 ) 2>&1 | grep real
real 0m1.028s
```

Output in messages.log:

```
$ docker exec -it $(docker ps -f "ancestor=websphere-liberty-plus" --format "{{.ID}}") tail -f /logs/messages.log
requestID=AAACFLR3zVr_AAAAAAAAAAA # eventType=websphere.servlet.service #
contextInfo=swat | Sleep?duration=1000 # duration=1017.062ms
```

Page 472
Docker Compose

Docker compose is a simple way to build multiple containers together:
https://docs.docker.com/compose/compose-file/

Example WAS-based compositions: https://github.com/kgibm/websphere_docker_examples

docker-compose.yml

The docker-compose.yml file describes how to build and wire together multiple containers. In the following example, there is a container that's logically called "app" with a Dockerfile in the "./was" sub-folder and this type of container will expose port 9443 to other containers, and there is a container that's logically called "proxy" with a Dockerfile in the "./nginx" sub-folder and this type of container will expose port 80 to the host, and it "depends_on" the "app" container(s) so that proxy is only started after app:

```yaml
version: "3"
services:
  app:
    build:
      context: was
      dockerfile: Dockerfile
    expose:
      - "9443"
    environment:
      - ENABLE_BASIC_LOGGING=true
  proxy:
    build:
      context: nginx
      dockerfile: Dockerfile
    ports:
      - "80:80"
    depends_on:
      - app
```

docker-compose build

The $(docker-compose build) command builds all of the container Dockerfiles in the docker-compose.yml file. This should be run any time any of the Dockerfiles are changed.

docker-compose up

The $(docker-compose up) command starts all of the containers in the docker-compose.yml file. Multiple instances of a particular service may be started. For example:

```bash
docker-compose up --scale app=2
```

Use the -d parameter to start everything in the background. For example:

```bash
docker-compose up --scale app=2 -d
```
**docker-compose logs**

If starting in the background, tail the logs after starting:

```bash
docker-compose logs -t -f
```

**docker-compose down**

The `$(docker-compose down)` command stops all of the containers in the docker-compose.yml file. This is normally run after typing Ctrl^C in the `$(docker-compose up)` window.

**Kubernetes**

Kubernetes (also known as k8s or kube) is an open-source system for automating deployment, scaling, and management of containerized applications: [https://kubernetes.io/](https://kubernetes.io/)

Book on Kubernetes written by IBMers: [https://www.ibm.com/downloads/cas/YXQPBAB3](https://www.ibm.com/downloads/cas/YXQPBAB3)

**IBM Cloud**

IBM Cloud Background: [https://www.ibm.com/cloud/](https://www.ibm.com/cloud/)

Login: [https://cloud.ibm.com/](https://cloud.ibm.com/)

**IBM Cloud Private**

IBM Cloud Private (ICP) is an on-premise cloud based on Kubernetes and Cloud Foundry: [https://www.ibm.com/cloud/private](https://www.ibm.com/cloud/private)

IBM Cloud Paks are pre-packaged Kubernetes enterprise capabilities for deployment, lifecycle management, and production use cases: [https://www.ibm.com/cloud/private/why-containers](https://www.ibm.com/cloud/private/why-containers)

IBM Helm charts for ICP: [https://github.com/IBM/charts](https://github.com/IBM/charts)

**WebSphere Liberty in ICP**


**Traditional WebSphere Application Server Base in ICP**

Traditional WebSphere Application Server Network Deployment in ICP using VMs


Transformation Advisor

Transformation Advisor is a free tool that provides a visual report of each application’s ability to move into a cloud-native environment: https://developer.ibm.com/recipes/tutorials/using-the-transformation-advisor-on-ibm-cloud-private/

IBM Cloud Kubernetes Service Diagnostics and Debug Tool

ibmcloud-iks-debug is a Helm chart which runs tests and gathers information from an IKS cluster to help debug and resolve issues: https://cloud.ibm.com/containers-kubernetes/solutions/helm-charts/ibm.ibmcloud-iks-debug


Hazelcast

The Hazelcast Helm chart may be used for functions such as HTTP session caching: https://github.com/IBM/charts/tree/master/community/hazelcast-enterprise

StockTrader Sample Application

- https://hub.docker.com/u/ibmstocktrader

IBM Cloud Application Platform

IBM Cloud Application Platform is a package which includes IBM Cloud Private, Traditional WAS, WAS Liberty and more, and provides a flexible path to modernize applications into the cloud: https://www.ibm.com/us-en/marketplace/ibm-cloud-app-platform-for-hybrid-deploy

IBM Cloud Automation Manager

IBM Cloud Automation Manager (CAM) is an ICP service using Terraform, Chef and Helm charts for automating application deployments into one or more clouds: https://www.ibm.com/us-en/marketplace/cognitive-automation

IBM Multicloud Manager

Manage multiple on-premise and/or off-premise clouds: https://www.ibm.com/cloud/multicloud-manager

IBM Cloud Garage

IBM Cloud Garages are physical locations with IBM experts who help customers move to the cloud
and create reference architectures and examples: [https://www.ibm.com/cloud/garage/](https://www.ibm.com/cloud/garage/)


## Virtualization

### Virtualization Recipe

1. Do not overcommit memory.
2. Use hypervisor utilities to monitor resource utilizations in addition to guest utilities.
3. When overcommitting CPU, take care just as you would when running multiple processes on the same physical CPU.
4. If using geographically separated data centers, measure cross-data center latencies.

### Key Concepts

Virtualization is an abstraction or a masking of underlying physical resources (such as a server) from operating system images or instances running on the physical resource. By abstracting the operating system from the underlying hardware, you can create multiple independent or isolated OS environments on a given set of hardware and, depending on the virtualization technology in use, the OS environments can either be homogenous or heterogeneous. This capability enables the consolidation of multiple environments on a single server that are dedicated and isolated from other environments.

Application virtualization... addresses application level workload, response time, and application isolation within a shared environment. A prominent example of an application virtualization technology is WebSphere Virtual Enterprise ([Intelligent Management](https://www.ibm.com/cloud/garage/architectures/application-modernization/reference-architecture)).

Server virtualization enables the consolidation of physical multiple servers into virtual servers all running on a single physical server, improving the resource utilization while still not exceeding capacity. Additional benefits of server virtualization include savings in power, cooling, and floor space, and probably lower administrative costs as well.


The virtualization system is called the hypervisor or host, and the virtualized system running on top of the hypervisor is called the guest. "A hypervisor can be classified into two types: Type 1, also known as "native" or "bare metal," where the hypervisor is the operating system or it's integral to the operating system. Examples of type 1 hypervisors would be VMware ESX and IBM PowerVM™ to name but two. Type 2 refers to "hosted" or "software applications," where the hypervisor is an application running on the operating system. Some examples include VMware Server, VMware Workstation, and Microsoft® Virtual Server." [http://www.ibm.com/developerworks/websphere/techjournal/1102_webcon/1102_webcon.html](http://www.ibm.com/developerworks/websphere/techjournal/1102_webcon/1102_webcon.html)
On recent versions of the IBM JVM, if you have very short-lived applications in a dynamic, cloud-like environment and you're experiencing performance problems, consider using the option -Xtune:virtualized (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/appendices/cmdline/Xtunevirtualized.html).

Sometimes it is difficult to prove whether or not a guest is affected by other guests. If possible, move or duplicate the guest to a similarly sized host with little or no other guest activity to test this hypothesis.

In general, hypervisor resource statistics (e.g. CPUs, memory, etc.) are more accurate than guest statistics.

While CPU over-provisioning may be tolerable, memory over-provisioning, particularly with Java applications, is not recommended.

Consider dedicating memory for virtual machines and, in general, avoid spanning CPU sockets.

Ensure sufficient physical resources for the hypervisor itself (e.g. CPUs).

**Guest Mobility**

Technologies such as Power's Live Partition Mobility and VMWare's vMotion can dynamically move guests between hosts while running and performing work. This isn't magic and it involves pausing the guest completely during the move. In addition, workloads with a high rate of memory references may have continuing effects after the pause due to memory cache hit rates. Other variables may also come into play such as the distance of host-to-host communications increasing due to the change (e.g. if the network distance increases, or if two hosts shared a CPU chip or NUMA interconnects and then one moved away, etc.).

Depending on the duration of the pause, guest mobility may be acceptable similar to a full garbage collection, or it may be unacceptable similar to memory thrashing or excessive CPU overcommit. In general, the use of these technologies should be minimized for production workloads and tested extensively to make sure the pauses and response time degradation are acceptable in the context of service level requirements. Internal IBM tests have shown that there may be workload pauses and throughput decreases associated with a guest move, which vary based on the factors mentioned above and may or may not be acceptable for workloads with high service levels.

**VMWare**

Consider for a moment the number of idle or under-utilized servers that might exist in a typical lab or data center. Each of these systems consumes power, rack space, and time in the form of maintenance and administration overhead. While it is costly to allow servers to remain idle, it's also unreasonable in most cases to power a system down. Consolidation through virtualization provides a solution by pooling hardware resources and scheduling them according to demand. If a VM has idle resources, they can be redirected to other systems where they are needed. Under this model the cost of idle servers can be minimized, while allowing their function to continue.

Various scenarios were measured to demonstrate the performance and scalability of WebSphere Application Server V8.5.5.1 within VMware ESXi 5.5 VMs as compared to on-
the-metal (OTM) results on state-of-the-art multi-core hardware. ESXi performance of a typical WebSphere Application Server application was generally within ~15% of OTM when running on an unsaturated system.

Do not over commit memory for WebSphere Application Server V8.5.5.1 VM deployments. It is critical for the host to have enough physical memory for all the VMs. Over committing memory in this scenario can result in drastic performance problems.

Over committing CPU can improve both density and performance if the ESXi host is not saturated. However, if the host is saturated then this could result in an incremental performance loss. Response times steadily increase when all CPUs are heavily loaded.

OS level performance statistics within a VM are not accurate. Do not rely on these statistics for tuning/management. ESX provides accurate statistics at the hypervisor level.

To achieve the optimal configuration, single Instance VMs should not span socket boundaries... If a single VM has more vCPUs than can fit within a single socket, consider vertical scaling the VMs for better performance. If a VM needs more vCPUs than can fit inside a single socket, then it is recommended to configure the VM with virtual sockets that match the underlying physical sockets architecture.

ftp://public.dhe.ibm.com/software/webservers/appserv/was/WASV8551_VMware_performance_2_17.pdf

vMotion

VMware has the ability to perform "live migrations" which "allows you to move an entire running virtual machine from one physical server to another, with no downtime." (see https://www.vmware.com/products/vsphere/vmotion.html) However, the actual movement of the running virtual machine can affect the virtual machine's performance especially if the virtual machine is moved frequently.

Performance Best Practices for VMware:
http://www.vmware.com/pdf/Perf_Best_Practices_vSphere5.5.pdf


Review the virtual CPU to physical CPU mapping. In some cases, a virtual CPU may be a CPU core thread rather than a CPU core. Review the Operating Systems chapter for background on CPU allocation.

Large Pages

Using large pages improves overall SPECjbb2005 performance by 8-10 percent... [which] comes from a significant reduction in L1 DTLB misses... ESX Server 3.5 and ESX Server 3i v3.5 enable large page support by default. When a virtual machine requests a large page,
the ESX Server kernel tries to find a free machine large page.

When free machine memory is low and before swapping happens, the ESX Server kernel attempts to share identical small pages even if they are parts of large pages. As a result, the candidate large pages on the host machine are broken into small pages. In rare cases, you might experience performance issues with large pages. If this happens, you can disable large page support for the entire ESX Server host or for the individual virtual machine.


Balooning

The memory balloon driver (vmmemctl) collaborates with the server to reclaim pages that are considered least valuable by the guest operating system. The driver uses a proprietary ballooning technique that provides predictable performance that closely matches the behavior of a native system under similar memory constraints. This technique increases or decreases memory pressure on the guest operating system, causing the guest to use its own native memory management algorithms. When memory is tight, the guest operating system determines which pages to reclaim and, if necessary, swaps them to its own virtual disk.

If necessary, you can limit the amount of memory vmmemctl reclaims by setting the sched.mem.maxmemctl parameter for a specific virtual machine. This option specifies the maximum amount of memory that can be reclaimed from a virtual machine in megabytes (MB).


This has some known issues on Linux: http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1003586

On Linux, if the sum of processes’ resident memory is significantly less than the total memory used (whether from free, top, or meminfo) - i.e. memory used minus filecache, minus buffers, minus slab - then this may be ballooning. There have been cases where ballooning can cause runaway paging and spark the OOM killer.

How to find out what amount of memory a VMWare balloon driver has consumed from a virtualized server: https://access.redhat.com/site/solutions/445113

esxtop

Guest Operating Systems

Virtualized Linux

The vmstat command includes an "st" column that reports CPU time "stolen" from the guest: "st: Time stolen from a virtual machine. Prior to Linux 2.6.11, unknown." (http://man7.org/linux/man-pages/man8/vmstat.8.html). This is also available in the top command.

Cloud

Key Concepts

1. Virtualization by itself does not increase capacity. You still have a finite amount of resources; i.e. CPU, memory, network, disks, etc.
   - Virtualization may allow you to better, and more effectively, use those resources.
   - You will incur some overhead for the hypervisor.
   - The consequences of over committing memory are significantly more dramatic than that of CPU resources
     - For example, in the PureApplication Server environment, over committing memory is not allowed
2. Other tuning concepts outlined in this Cookbook should also be adhered to when running in a virtual environment including
   - Operating System
   - Java
   - Linux
   - Database
3. Depending on your runtime environment, virtualization may provide you with the ability to auto scale your workload(s) based on policy(s) and demand, for example:
   - PureApplication Server
   - SoftLayer

4. Disk drive capacity has been increasing substantially over the past several years. It is not unusual to see disk drives with storage capacity from 500 Megabytes to 3 Terabytes or more. However, while storage capacity has certainly increased, IOPS (Input-output Operations Per Second) has not come close to keeping pace, particularly for Hard Disk Drives (HDD's). The nature of virtualization is to try to pack as many VM's (density) as possible on a physical compute node. Particular attention should be given to the IOPS requirements of these VM's, and not just their disk storage requirements. Newer disk technology's, like Solid State Drives (SSD's) and Flash drives, offer significant IOPS improvements, but may, or may not, be available in your environment. Some environments are connected to SANs (a network of drives) which can introduce latency to a virtual machine and must be monitored to ensure that disk I/O time is acceptable for the virtual machine. Any lag in the disk I/O latency can affect applications running in the virtual machine. Applications that tend to log a lot of data (error, info, audit, etc) can suffer performance issues if the latency is too high.

**Trends**

1. The cost of memory outweighs the cost of CPU, disk, and network resources in cloud environments. This is pushing many customers to reduce memory usage and increase CPU usage.
2. Various services are starting to be provided as pay-per-use API calls. This is pushing many customers to cache the results of expensive API calls.

**Scalability and Elasticity**

Scalability and elasticity for virtual application patterns in IBM PureApplication System:

**Databases**

Here is a list of databases that are fully tested & supported with WAS:

Terms:

- **Cardinality**: with respect to tables, the number of rows in the table. With respect to indexed columns, the number of distinct values of that column in a table.
- **Normalization** is the process of restructuring a data model by reducing its relations to their simplest forms. It is a key step in the task of building a logical relational database design. Normalization reduces redundancy from your data and can improve the performance of update and delete statements, since you only have to do it in one place. By normalizing your data, you try to ensure that all columns in the table depend on the primary key. The disadvantage of a
fully normalized data structure is in data retrieval operations, specifically when a query is accessing a large number of related pieces of data from different tables via join operations. For more information about Normalization, author C.J. Date is one of the better resources.

- Denormalization is the intentional duplication of columns in multiple tables whose consequence is increased data redundancy. Denormalization is sometimes necessary to minimize performance problems and is a key step in designing a physical relational database design.

**IBM DB2**

Display configuration: `db2 get db cfg`

DB2 Self-tuning: `db2 autoconfigure apply db and dbm`


Located in the DB2 Control Center, the DB2 configuration advisor calculates and displays recommended values for the DB2 buffer pool size, the database, and the database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to establish clones, set the MaxAppls value as the maximum number of connections multiplied by the number of clones. The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources, set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

For example, MaxAppls = (number of connections set for the data source + number of connections in the session manager) multiplied by the number of clones.

After calculating the MaxAppls settings for the WebSphere Application Server database and each of the application databases, verify that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls values. For example, MaxAgents = sum of MaxAppls for all databases.

For systems with multiple hard disk drives, you can gain large performance improvements by setting the log files for each database on a different hard drive from the database files.

How to view or set: At a DB2 command prompt, issue the command: `db2 update db cfg for [database_name] using newlogpath [fully_qualified_path]`.

Recommended value: Use a separate high-speed drive, preferably performance enhanced through a redundant array of independent disk (RAID) configuration.

If lock escalations are causing performance concerns, you might need to increase the value
of [maxlocks] or the locklist parameter... You can use the database system monitor to determine if lock escalations are occurring.


- Maintain current indexes on tables: utilize the DB2 Design Advisor (available from the DB2 Control Center, or command line) to help determine indexes that could improve performance.
- Update catalog statistics: DB2 uses these to optimize access to key tables. The easiest way to maintain statistics is via the DB2 Automatic Table Maintenance feature, which runs the RUNSTATS command in the background as required to ensure that the correct statistics are collected and maintained. By default, this feature is not enabled. It may be turned on from the DB2 Control Center.
- Set buffer pool size correctly: a buffer pool is an area of memory into which database pages are read, modified, and held during processing; accessing pages from the buffer pool is much faster than accessing pages from physical devices. To choose appropriate buffer pool size settings, monitor database container I/O activity, by using system tools or by using DB2 buffer pool snapshots. Be careful to avoid configuring large buffer pool size settings which lead to paging activity on the system.

https://w3quickplace.lotus.com/QuickPlace/wasperf/PageLibrary852569AF00670F15.nsf/DefaultView/1CCEB50DD9A9C561852576030042A65C/$File/WebSphere%20BPM%206.2%20How%20To%20Win%20Performance%20POCs.pdf?OpenElement

Put frequently updated columns together and at the end of the row. This has an effect on update performance due to the following logging considerations: For fixed length row updates, DB2 logs from the first changed column to the last changed column. For variable length row updates, DB2 logs from the first changed byte to the end of the row. If the length of a variable length column changes, this will result in a change to the row header (which includes the row length), and thus the entire row will be logged.

**Query Execution Times**

To get per-query execution times, create a DB2 event monitor (note on the create event monitor command, single quotes around the path are required):

```
$ mkdir $PATH
$ chmod 777 $PATH
$ db2 connect to <db_name> user <inst_user> using <password>
$ db2 "create event monitor $NAME for statements write to file '$PATH'
$ db2 "set event monitor $NAME state 1"
```

To disable an event monitor:

```
$ db2 "set event monitor $NAME state 0"
```

To process event monitor data to a human readable form:
$ db2evmon -path $PATH > commands.out

To list all event monitors:
$ db2 "select * from SYSCAT.EVENTMONITORS"

To completely delete an event monitor:
$ db2 "drop event monitor $NAME"

Example of a single query execution from db2evmon output:

Statement Event ...
  Text     : select id,symbol from MYTABLE
  Start Time: 02-09-2010 18:21:46.159875
  Stop Time: 02-09-2010 18:21:46.164743
  Exec Time: 0.004868 seconds...

Tables

A tablespace is a physical storage object that provides a level of indirection between a database and the tables stored within the database. It is made up of a collection of containers into which database objects are stored. A container is an allocation of space to a table space. Depending on the table space type, the container can be a directory, device, or file.

System Managed Space (SMS): stores data in operating system files. They are an excellent choice for general purposes use. They provide good performance with little administration cost.

Database Managed Space (DMS): with database-managed space (DMS) table spaces, the database manager controls the storage space.

DMS tablespaces usually perform better than SMS tablespaces because they are pre-allocated and do not have to spend time extending files when new rows are added. DMS tablespaces can be either raw devices or file system files. DMS tablespaces in raw device containers provide the best performance because double buffering does not occur. Double buffering, which occurs when data is buffered first at the database manager level and then at the file system level, might be an additional cost for file containers or SMS table spaces.

If you use SMS tablespaces, consider using the db2empfa command on your database. The db2empfa (Enable Multipage File Allocation) tool enables the use of multipage file allocation for a database. With multipage file allocation enabled for SMS table spaces, disk space is allocated one extent rather than one page at a time, improving INSERT throughput.

$ db2 "LIST TABLESPACES SHOW DETAIL"

Buffer Pools

There is no definitive answer to the question of how much memory you should dedicate to the buffer pool. Generally, more is better. A good rule of thumb would be to start with about 75% of your system’s main memory devoted to buffer pool(s), but this rule is applicable only if the machine is a dedicated database server.
If your tablespaces have multiple page sizes, then you should create one buffer pool for each page size.

Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. Data is accessed much faster from memory than from disk.

How to view or set: To view the current value of buffpage for database x, issue the DB2 command get db cfg for x and look for the value BUFFPAGE. To set BUFFPAGE to a value of n, issue the DB2 command update db cfg for x using BUFFPAGE n and set NPAGES to -1 as follows:

```db2
-- go to DB2 command mode, otherwise the following "select" does not work as is
connect to x  <-- (where x is the particular DB2 database name)
select * from syscat.bufferpools
   (and note the name of the default, perhaps: IBMDEFAULTBP)
   (if NPAGES is already -1, there is no need to issue following command)
alter bufferpool IBMDEFAULTBP size -1
   (re-issue the above "select" and NPAGES now equals -1)
```

You can collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio as follows:

Collect the snapshot:

- Issue the update monitor switches using bufferpool on command.
- Make sure that bufferpool monitoring is on by issuing the get monitor switches command.
- Clear the monitor counters with the reset monitor all command.
- Run the application.
- Issue the get snapshot for all databases command prior to all applications disconnect from the database, otherwise statistics are lost.
- Issue the update monitor switches using bufferpool off command.
- Calculate the hit ratio by looking at the following database snapshot statistics:
  - Buffer pool data logical reads
  - Buffer pool data physical reads
  - Buffer pool index logical reads
  - Buffer pool index physical reads
- Default value: 250
- Recommended value: Continue increasing the value until the snapshot shows a satisfactory hit rate.

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk to service a page request. That is, the page is already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk input and output. Calculate the buffer pool hit ratio as follows:

\[
P = \text{buffer pool data physical reads + buffer pool index physical reads}
\]
\[
L = \text{buffer pool data logical reads + buffer pool index logical reads}
\]
\[
\text{Hit ratio} = \left(1 - \frac{P}{L}\right) \times 100\%
\]
Indexing

An index is a set of keys, each pointing to a row, or rows in a table. An index serves to ensure uniqueness, as in the case of Primary Key, and to allow more efficient access to rows in a table by creating a direct path to the data through pointers. The SQL optimizer automatically chooses the most efficient way to access data in tables. The optimizer takes indexes into consideration when determining the fastest access path to data.

An index will impact disk storage usage, insert and delete processing, and database maintenance.

The intent of a clustering index is so that the sequence of key values closely corresponds to the sequence of rows stored in a table.

Create as few indexes as possible. Consider creating the INDEXES with the “ALLOW REVERSE SCANS” option. Pay close attention to the order of the columns in the index. Don’t create redundant indexes. Use DB2 “Explain” facilities to determine the actual usage of the indexes.

Logging

One of the main purposes of all database systems is to maintain the integrity of your data. All databases maintain log files that keep records of database changes. DB2 logging consists of a set of primary and secondary log files that contain log records that record all changes to a database. The database log is used to roll back changes for units of work that are not committed and to recover a database to a consistent state. DB2 provides two logging strategy choices.

Circular logging is the default log mode. With circular logging, the log records fill the log files and then overwrite the initial log records in the initial log file. The overwritten log records are not recoverable. This type of logging is typically not suited for a production application.

Log Retain logging is a setting where a log is archived when it fills with log records. New log files are made available for log records. Retaining log files enables roll-forward recovery. Roll-forward recovery reappplies changes to the database based on completed units of work (transactions) that are recorded in the log. You can specify that roll-forward recovery is to the end of the logs, or to a particular point in time before the end of the logs. Archived log files are never directly deleted by DB2, therefore, it is the applications’ responsibility to maintain them; i.e. archive, purge, etc.

Placement of the log files needs to be optimized, not only for write performance, but also for read performance, because the database manager will need to read the log files during database recovery.

Increase the size of the database configuration Log Buffer parameter (logbufsz). This parameter specifies the amount of the database heap to use as a buffer for log records before writing these records to disk.

Buffering the log records will result in more efficient logging file I/O because the log records will be written to disk less frequently, and more log records will be written at each time.
Reorg

SQL statement performance can deteriorate after many updates, deletes or inserts.

Use the DB2 reorgchk update statistics on table all command to perform the runstats operation on all user and system tables for the database to which you are currently connected. Rebind packages using the bind command. If statistics are available, issue the db2 -v "select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes" command on DB2 CLP. If no statistic updates exist, nleaf and nlevels are -1, and stats_time has an empty entry (for example: "-"). If the runstats command was previously run, the real-time stamp from completion of the runstats operation also displays under stats_time. If you think the time shown for the previous runstats operation is too old, run the runstats command again.

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rprf_db2parameters.html

Runstats

The DB2 optimizer uses information and statistics in the DB2 catalog in order to determine the best access to the database based on the query provided. Statistical information is collected for specific tables and indexes in the local database when you execute the RUNSTATS utility. When significant numbers of table rows are added or removed, or if data in columns for which you collect statistics is updated, execute RUNSTATS again to update the statistics.

After running RUNSTATS on your database tables, you need to rebind your applications to take advantage of those new statistics. This is done to ensure the best access plan is being used for your SQL statements. To clear the contents of the SQL cache, use the FLUSH PACKAGE CACHE sql statement.

Explain

Explain allows you to capture information about the access plan chosen by the optimizer as well as performance information that helps you tune queries. Before you can capture explain information, you need to create the relational tables in which the optimizer stores the explain information and you set the special registers that determine what kind of explain information is captured.

- db2 ttf EXPLAIN.DDL (located in sqllib/misc directory)
- db2exfmt" this command line tool is used to display explain information in preformatted output.

db2expln and dynexpln: these command line tools are used to see the access plan information available for one or more packages of static SQL statements. Db2expln shows the actual implementation of the chosen access plan. It does not show optimizer information. The dynexpln tool, which uses db2expln within it, provides a quick way to explain dynamic SQL statements that contain no parameter markers. This use of db2expln from within dynexpln is done by transforming the input SQL statement into a static statement within a pseudo-package. When this occurs, the information may not always be completely accurate. If complete accuracy is desired, use the explain facility. The db2expln tool does provide a relatively compact and English-like overview of what operations will occur at run-time by examining the actual access plan generated.
**Isolation Levels**

An isolation level determines how data is locked or isolated from other processes while the data is being accessed. The isolation level will be in effect for the duration of the unit of work. DB2 supports the following isolation levels, listed in order of most restrictive to least restrictive:

1. **Repeatable Read** - An isolation level that locks all the rows in an application that are referenced within a transaction. When a program uses repeatable read protection, rows referenced by the program cannot be changed by other programs until the program ends the current transaction.

2. **Read Stability** - An isolation level that locks only the rows that an application retrieves within a transaction. Read stability ensures that any qualifying row that is read during a transaction is not changed by other application processes until the transaction is completed, and that any row changed by another application process is not read until the change is committed by that process.

3. **Cursor Stability** - An isolation level that locks any row accessed by a transaction of an application while the cursor is positioned on the row. The lock remains in effect until the next row is fetched or the transaction is terminated. If any data is changed in a row, the lock is held until the change is committed to the database.

4. **Uncommitted Read** - An isolation level that allows an application to access uncommitted changes of other transactions. The application does not lock other applications out of the row that it is reading, unless the other application attempts to drop or alter the table. Sometimes referred to as “Dirty Reads”

**Lock Timeouts**

To view the current value of the lock timeout property for database xxxxxx, issue the DB2 get db cfg for xxxxxx command and look for the value LOCKTIMEOUT. To set LOCKTIMEOUT to a value of n, issue the DB2 update db cfg for xxxxxx command using LOCKTIMEOUT n, where xxxxxx is the name of the application database and n is a value between 0 and 30 000 inclusive.

Default value: -1, meaning lock timeout detection is turned off. In this situation, an application waits for a lock if one is not available at the time of the request, until either the lock is granted or a deadlock occurs.

Recommended value: If your database access pattern tends toward a majority of writes, set this value so that it gives you early warning when a timeout occurs. A setting of 30 seconds suits this purpose. If your pattern tends toward a majority of reads, either accept the default lock timeout value, or set the property to a value greater than 30 seconds.

https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/rprf_db2parameters.html

If lock escalations occur frequently, increase the value of either locklist or maxlocks, or both.

**Query Tuning**

Use the OPTIMIZE FOR n ROWS clause to give priority to retrieving the first n rows in the full result
Use the FETCH FIRST n ROWS ONLY clause to retrieve only a specified number of rows. Take advantage of row blocking, by specifying the FOR READ ONLY, FOR FETCH ONLY, OPTIMIZE FOR n ROWS clause, or if you declare your cursor as SCROLLing. This will improve performance, and, in addition, improve concurrency because exclusive locks are never held on the rows retrieved.

Consider the use of APPEND MODE

Insert multiple rows with one INSERT statement

**Disk**

A database that would have taken 36 * 1 GB drives a number of years ago can now be placed on one disk. This highlights the database I/O problems. For example, if each 1 GB disk drive can do 80 I/O operations a second, this means the system can do a combined 36 * 80 = 2880 I/O operations per second. But a single 36 GB drive with a seek time of 7 ms can do only 140 I/O operations per second. While increased disk drive capacity is good news, the lower numbers of disks cannot deliver the same I/O throughput.

**DB2 Configuration**

Number of asynchronous page cleaners (NUM_IOCLEANERS) - This parameter controls the number of page cleaners that write changed pages from the buffer pool to disk. You may want to increase this to the number of physical disk drive devices you have. The default is 1.

Enable intra-partition parallelism (INTRA_PARALLEL) – if you have a multi-processor SMP system, setting this parameter to YES may improve performance. The default is NO.

To optimize for INSERT speed at the possible expense of faster table growth, set the DB2MAXFSCRSEARCH registry variable to a small number. To optimize for space reuse at the possible expense of INSERT speed, set DB2MAXFSCRSEARCH to a larger number.

**Snapshots**

Collecting performance data introduces overhead on the operation of the database. DB2 provides monitor switches to control which information is collected. You can turn these switches on by using the following DB2 commands:

- UPDATE MONITOR SWITCHES USING BUFFERPOOL ON ;
- UPDATE MONITOR SWITCHES USING LOCK ON ;
- UPDATE MONITOR SWITCHES USING SORT ON ;
- UPDATE MONITOR SWITCHES USING STATEMENT ON ;
- UPDATE MONITOR SWITCHES USING TABLE ON ;
- UPDATE MONITOR SWITCHES USING UOW ON ;

You can access the data that the database manager maintains either by taking a snapshot or by using an event monitor.

Use the GET SNAPSHOT command to collect status information and format the output for your use.
Some of the most useful options are:

- **GET SNAPSHOT FOR DATABASE** - Provides general statistics for one or more active databases on the current database partition.
- **GET SNAPSHOT FOR APPLICATIONS** - Provides information about one or more active applications that are connected to a database on the current database partition.
- **GET SNAPSHOT FOR DATABASE MANAGER** - Provides statistics for the active database manager instance.
- **GET SNAPSHOT FOR LOCKS** - Provides information about every lock held by one or more applications connected to a specified database.
- **GET SNAPSHOT FOR BUFFERPOOLS** - Provides information about buffer pool activity for the specified database.
- **GET SNAPSHOT FOR DYNAMIC SQL** - Returns a point-in-time picture of the contents of the SQL statement cache for the database.

**db2batch**

A benchmark tool called db2batch is provided in the sqllib/bin subdirectory of your DB2 installation. This tool can read SQL statements from either a flat file or standard input, dynamically describe and prepare the statements, and return an answer set.

**IBM DB2 for z/OS**

"First, ensure that your DB2 logs are large enough, are allocated on the fastest volumes you have, and make sure they have optimal CI sizes.

Next, ensure that you have tuned your bufferpools so that the most often-read data is in memory as much as possible. Use ESTOR and hyperpools.

You may want to consider pre-formatting tables that are going to be heavily used. This avoids formatting at runtime.

Ensuring DB2 Tracing Under the DB2 for z/OS Universal Driver is Turned Off:

If the db2.jcc.propertiesFile jvm property has been defined to specify a DB2 jcc properties file to the WebSphere Application Server for z/OS, ensure that the following trace statements in the file are commented out if they are specified:

```
# jcc.override.traceFile=<file name>
# jcc.override.traceFile=<file name>
```

If any of the DB2 Universal JDBC Driver datasources your applications are using are defined with a nonzero traceLevel custom property, use the WebSphere Application Server for z/OS Administrative console to set the traceLevel to zero.

Be sure to define indexes on all your object primary keys. Failure to do so will result in costly tablespace scans.
Ensure that, once your tables are sufficiently populated, you do a re-org to compact the tables. Running RUNSTATS will ensure that the DB2 catalog statistics about table and column sizes and accesses are most current so that the best access patterns are chosen by the optimizer.

Enable dynamic statement caching in DB2. To do this, modify your ZPARMS to say CACHEDYN(YES) MAXKEEPD(16K). Depending on the application, this can make a very significant improvement in DB2 performance. Specifically, it can help JDBC and LDAP query.

Increase DB2 checkpoint interval settings to a large value. To do this, modify your ZPARMS to include CHKFREQ=xxxxx, where xxxxx is set at a high value when doing benchmarks (e.g. CHKFREQ=16000000). On production systems there are other valid reasons to keep checkpoint frequencies lower, however.


**DB2 on Linux on Power**

**Write I/O to the Transaction Log**

Before a transaction is committed, it must be written to the transaction log. This can become a primary bottleneck. This can be lessened by isolating transaction logs.

**Data Compression**

If there is available CPU and I/O is the bottleneck, consider data compression with the DB2 Storage Optimization feature.

```
alter table <table_name> compress yes
alter index <index_name> compress yes
reorg table <table_name> RESETDICTIONARY
reorg indexes all for table <table_name>
runstats on table <table_name> with distribution and detailed indexes all allow read access
```

**DB2 JCC Type 2 Driver**

As of February, 2018, the default timerLevelForQueryTimeOut of QUERYTIMEOUT_STATEMENT_LEVEL (1) creates a timer object for each statement execution (when there is a non-zero timeout) which may have a large performance impact. The alternative of creating a single timeout object for each connection (e.g. with -Ddb2.jcc.override.timerLevelForQueryTimeOut=2) in one case improved performance by 65%. Changing the default means holding the timer and related memory for longer for each connection in a connection pool, but this is an acceptable cost for most customers for the improved performance. A feature request has been opened to change the default.
Oracle Database

Review the Oracle Database (software and hardware) tuning in the latest SPECjEnterprise results submitted by Oracle:


- **Update Database Statistics:** statistics are maintained on tables and indexes. Updating statistics allows the query optimizer to create better performing access plans for evaluating queries. One approach to manually updating statistics on all tables in a schema is to use the dbms_stats utility:

  ```sql
  execute dbms_stats.gather_schema_stats( -
  ownname => 'your_schema_name', -
  options => 'GATHER AUTO', -
  estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, -
  cascade => TRUE, -
  method_opt => 'FOR ALL COLUMNS SIZE AUTO', -
  degree => 15);
  ```

- **Set Buffer Cache sizes correctly:** this reference discusses this issue in detail: [https://docs.oracle.com/cd/B19306_01/server.102/b14211/memory.htm#g77696](https://docs.oracle.com/cd/B19306_01/server.102/b14211/memory.htm#g77696)

- **Set Log Files Appropriately:** Unlike DB2, Oracle performs an expensive checkpoint operation when switching logs. The checkpoint involves writing all dirty pages in the buffer cache to disk. Therefore, it is important to make the log files large enough that switching occurs infrequently. Also, applications which generate a high volume of log traffic need larger log files to achieve this goal.

- **Maintain proper table indexing:** a database environment that requires additional indexes will often exhibit performance degradation over time; in some cases the performance degradation can be profound. Environments that need additional indexes often exhibit heavy read I/O on devices holding the tablespace datafiles. To assist in determining which additional indexes could improve performance, Oracle 10g provides the Automatic Database Diagnostic Monitor. It has the capability to help define and design indexes suitable for a particular workload.

- **When using the Oracle RAC product,** configure the database nodes as Active-Passive. This generally provides optimal system performance while also maintaining high availability via failover support.

The following references are useful:

- **Oracle 10g Release 2 documentation (includes a Performance Tuning Guide)** [http://www.oracle.com/pls/db102/homepage](http://www.oracle.com/pls/db102/homepage)
- [https://w3quickplace.lotus.com/QuickPlace/wasperf/PageLibrary852569AF00670F15.nsf/ Sdefaultview/1CCEB50DD9A9C561852576030042A65C/$File/WebSphere%20BPM %206.2%20How%20To%20Win%20Performance%20POCs.pdf?OpenElement](https://w3quickplace.lotus.com/QuickPlace/wasperf/PageLibrary852569AF00670F15.nsf/ Sdefaultview/1CCEB50DD9A9C561852576030042A65C/$File/WebSphere%20BPM %206.2%20How%20To%20Win%20Performance%20POCs.pdf?OpenElement)
The PROCESSES parameter is effectively equivalent to the maximum number of concurrent users plus the number of background processes.

The OPEN_CURSORS parameter value should be set high enough to prevent the application from running out of open cursors (handles to private SQL areas). For example, 3000.

The SESSION_CACHED_CURSORS parameter sets the number of cached closed cursors each session can have. For example, 1000.

The DB_FILES parameter specifies the maximum number of database files that can be opened for the database. For example, 3000.

The PRE_PAGE_SGA parameter determines whether Oracle reads the entire SGA into memory at instance startup. This setting can increase the amount of time necessary for instance startup, but it is likely to decrease the amount of time necessary for Oracle to reach its full performance capacity after startup.

The DB_WRITER_PROCESSES parameter can be set to take advantage of a multi-cpu system that modifies data heavily by enabling multiple DB writer processes. For example, use the formula

\[
\text{DB_WRITER_PROCESSES} = \frac{\text{CPU_COUNT}}{8}
\]

**Basic Commands**

List connected clients:

```
SELECT * FROM v$session
```

**Automatic Memory Management**

Automatic Memory Management (AMM) was introduced in Oracle 11g and allows most memory usage (SGA, PGA, buffer pools, shared pools, large pools, etc.) to be automatically sized (excluding the log buffer). For example:

1. Set a value for MEMORY_MAX_TARGET. Sufficient OS memory is required to support the value set. `MEMORY_MAX_TARGET=14464M`.

2. Set SGA_TARGET and PGA_AGGREGATE_TARGET to 0. If these values are nonzero then it defines the minimum size for the specified region.

3. Set MEMORY_TARGET to the total amount of memory you want to share between SGA and PGA. e.g. `MEMORY_TARGET=14464M`.

**Tibero Database**

Tibero is not tested with WAS. Presumably they are using a generic JDBC type 4 driver and so as long as they've written to the specification of JDBC/JCA, then WAS will support any connection pool issues; however, any issues with the database driver or the database are not supported.
**pureScale**

**DB2**

**WAS Workload Balancing and Automatic Client Reroute**

"Database applications running in a DB2 pureScale environment can use the DB2 transaction-level or connection-level workload balancing (WLB) functionality. WLB balances application requests among all members of the DB2 pureScale cluster. When WLB is enabled the DB2 clients distribute workload or application request based on the capacity (that is, the priority or weight) values in a server list that the DB2 pureScale server returns. These capacity values indicate the current load on a DB2 pureScale member. A member with a capacity value below that of the other members in the server list is busier than other members. [...]"

DB2 Java applications with the supported IBM Data Server Driver for JDBC and SQLJ (JCC driver) running in an IBM WebSphere Application Server environment can use the WLB or client affinities features to access databases on a DB2 pureScale cluster. [...]"

You must add JCC data source property enableSysplexWLB and set the value to true to enable WLB. [...]"

DB2 automatic client reroute (ACR) complements a continuously available DB2 pureScale cluster that offers 24/7 availability for clients connecting to mission-critical production systems. ACR is a feature in DB2 clients that takes application requests that are directed toward an offline DB2 pureScale member and reroutes them to active DB2 pureScale members. ACR is automatically enabled with WLB or client affinities so no additional steps are required to specify which member the application should connect to upon encountering an outage. [...]"


**Caching and WebSphere eXtreme Scale**

**Caching Recipes**

1. If available, enable the Java shared class and ahead-of-time compilation caches. WAS enables this by default, but you can increase the size if you have available memory. See the Java chapter.
2. Pre-compile Java Server Pages (JSPs). See the WAS chapter.
3. If possible, utilize the WAS Dynacache feature to cache servlet responses. See the HTTP section.
in the WAS chapter.

4. The application should set standardized response headers that indicate caching (e.g. Cache-Control in HTTP). See the Applications chapter.
   1. An alternative is to use a web server such as IHS to apply cache headers to responses based on rules. See the Web Servers chapter.

5. If possible, use the WebSphere eXtreme Scale (WXS) product to maximize data caching (see below).

6. Consider using an edge cache such as the WebSphere Caching Proxy. See the Web Servers chapter.

7. If using WebSphere Commerce, set Dynacache caches’ sharing modes to NOT_SHARED.

**General Caching Topics**

Caching (or lack thereof) can have dramatic performance impacts; however, caching must be carefully implemented to avoid inconsistent data:

Most Java EE application workloads have more read operations than write operations. Read operations require passing a request through several topology levels that consist of a frontend web server, the web container of an application server, the EJB container of an application server, and a database. WebSphere Application Server provides the ability to cache results at all levels of the network topology and Java EE programming model that include web services.

Application designers must consider caching when the application architecture is designed because caching integrates at most levels of the programming model. Caching is another reason to enforce the MVC pattern in applications. Combining caching and MVC can provide caching independent of the presentation technology and in cases where there is no presentation to the clients of the application.


In general, caches are held in memory or disk which must be properly sized for the additional cache usage. Caches may also introduce additional administration. Example caches (detailed in later chapters):

- Avoid your infrastructure altogether by telling the client (e.g. browser) to cache as much as possible with response headers.
- Cache whole or parts of responses (e.g. servlet caching).
- Use dedicated caching proxy servers between the end-user and application.
- Place static content as close to the user as possible (e.g. in a web server instead of the application server, content delivery networks, etc.).

**WebSphere eXtreme Scale (WXS)**

Caching is used to reduce execution path length at any layer to reduce the cost of each execution. This may lower the response time and/or lower the transaction cost. A grid is a set of maps that store data. Within a grid, partitions split maps across multiple container server JVMs using shards. A catalog
server coordinates shard placement and monitors container servers. There may be multiple catalog servers in a catalog service domain (which itself is a mini grid) for high availability. A partition has 1 primary shard and 0 or more replica shards. The primary shard receives the actual data (insert, update, remove). A replica shard may be either synchronous or asynchronous. If minSyncReplica is > 0, a transaction in a primary shard is only committed with the agreement of those replicas.

**Catalog Servers**

A catalog server references an objectGridServer.properties file. On WAS, this is often in <WAS>/properties and may be copied from <WAS>/optionalLibraries/ObjectGrid/properties/sampleServer.properties.

**Container Servers**


**Example objectGridDeployment.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?
<deploymentPolicy xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/deploymentPolicy../deploymentPolicy.xsd"
xmlns="http://ibm.com/ws/objectgrid/deploymentPolicy"
<objectgridDeployment objectgridName="grid1">
  <mapSet name="mapSet" numberOfPartitions="1" minSyncReplicas="0"
    maxSyncReplicas="0" developmentMode="true">
    <map ref="map1"/>
  </mapSet>
</objectgridDeployment>
</deploymentPolicy>
```

**WXS Client**


**Example objectGrid.xml**

```xml
<?xml version="1.0" encoding="UTF-8"?
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
xmlns="http://ibm.com/ws/objectgrid/config"
<objectGrids>
  <objectGrid name="grid1" txTimeout="120">
```

Page 496
Example code to put and get from a grid

```java
import com.ibm.websphere.objectgrid.ClientClusterContext;
import com.ibm.websphere.objectgrid.ConnectException;
import com.ibm.websphere.objectgrid.ObjectGrid;
import com.ibm.websphere.objectgrid.ObjectGridException;
import com.ibm.websphere.objectgrid.ObjectGridManagerFactory;
import com.ibm.websphere.objectgrid.ObjectMap;
import com.ibm.websphere.objectgrid.Session;
import com.ibm.websphere.objectgrid.plugins.TransactionCallbackException;

try {
    long key = 42;
    String value = "Hello World";
    ClientClusterContext ccc =
        ObjectGridManagerFactory.getObjectGridManager().connect("localhost:4809", null, null);
    ObjectGrid grid =
        ObjectGridManagerFactory.getObjectGridManager().getObjectGrid(ccc, "grid1");
    Session session = grid.getSession();
    ObjectMap map1 = session.getMap("map1");

    map1.setPutMode(ObjectMap.PutMode.UPSERT);
    map1.put(key, value);

    String fromGrid = (String) map1.get(key);
    System.out.println(fromGrid.equals(value));
} catch (ConnectException e) {
    throw new RuntimeException(e);
} catch (TransactionCallbackException e) {
    throw new RuntimeException(e);
} catch (ObjectGridException e) {
    throw new RuntimeException(e);
}
```

Best Practices

Have approximately 10 shards per container. So if you plan to have 50 containers for instance and you have one replica configured in your policy, we would recommend about 250 partitions. This allows for having extra shards available for adding containers in the future when you need to expand without taking a grid outage to change the number of partitions. With having extra partitions per container, elasticity can be achieved. The general formula is \((\text{number of containers} \times 10) / (1 + \text{number of replicas})\). That gives you the number of partitions to start with. That usually gives a whole number that is not prime. We recommend choosing a prime number that is close to the number that the formula returns.

When it comes to starting a lot of containers, we recommend making use of the xscmd commands of
suspendBalancing and resumeBalancing. You invoke suspendBalancing before staring the containers and resumeBalancing when you are complete. This approach allows eXtreme Scale to make one placement decision instead of multiple ones. If it was making a placement decision for each container as they start, the result can be a lot of unnecessary data movement.

Similarly when you are stopping containers and catalog servers, we recommend making use of the xscmd command of teardown to specify the servers you want to stop if you are stopping more than one. Again this approach allows you to limit the amount of data movement to be more efficient. There are filter options like host or zone to allow you to just say stop all containers on this host or in this zone for instance, or you can just give the complete list of the servers you want to stop. If you want to stop all containers, just run xscmd -c teardown without filters or a list of servers and it will stop all containers. If you want to stop all containers for a specific grid you can use the -g option to specify the grid to filter on.

Near Cache


The near cache is enabled by default for any map with a non-PESSIMISTIC lockStrategy (default OPTIMISTIC) (see the Spring section for an exception). It is also unbounded by default which may cause OutOfMemoryErrors if an evictor is not specified either through ttlEvictorType/timeToLive or a plugin evictor such as LRU through pluginCollectionRef. Alternatively, nearCacheInvalidationEnabled may be set to true to propagate invalidations from the grid to each nearCache: [http://www-01.ibm.com/support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/txsnearcache invalid.html?lang=en](http://www-01.ibm.com/support/knowledgecenter/SSTVLU_8.6.0/com.ibm.websphere.extremescale.doc/txsnearcache invalid.html?lang=en)

The increase in Java heap usage should be monitored to ensure the nearCache is not increasing the proportion of time in garbage collection too much (or its eviction/size should be tuned, or the heap increased).

If the map's copyMode is COPY_TO_BYTES or COPY_TO_BYTES_RAW, then nearCacheCopyMode should be set to NO_COPY, because any copying is unnecessary.

The near cache hit rate is a critical performance metric. A near cache occupancy may be limited by size (e.g. LRU/LFU evictor) or expired over time (e.g. TTL evictor).

Enable near cache statistics through the ObjectGrid Maps PMI module:
Performance Monitoring Infrastructure (PMI) > server1 > Custom monitoring level

Use this page to configure Performance Monitoring Infrastructure (PMI)

**Runtime** | **Configuration**

- server1
  - DCS Statistics
    - ExtensionRegistryStats name
    - Security Authentication
  - Security Authorization
  - agentManagerModule
  - Dynamic Caching
  - JDBC Connection Pools
  - HAManager
  - JVM Runtime
  - **ObjectGrid Maps**
    - ObjectGrids
    - Object Pool
    - odrStatModule

Enable | Disable

Select | Counter

- Batch update time for the loader.
- Map hit rate
- Number of bytes in use by this map
- Number of map entries
- Number of map gets
- Number of map hits

Then check the hit rate by analyzing hits / gets:

**Spring Integration**

WXS provides Spring integration for Spring >= 3.1: [http://www-](http://www-)

Page 499
Older documentation states that, generally, the nearCache is automatically enabled when the lockStrategy is NONE or OPTIMISTIC (default). This is true, except for the Spring provider which explicitly disables the nearCache even when it would have been enabled, unless a client override XML is provided (see CLIENT_OVERRIDE_XML in the link above).

Example Spring XML specifying the client override XML:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                       http://www.springframework.org/schema/beans/spring-beans.xsd
                       http://www.springframework.org/schema/tx
                       http://www.springframework.org/schema/tx/spring-tx.xsd">
  <bean id="domain"
        class="com.ibm.websphere.objectgrid.spring.ObjectGridCatalogServiceDomainBean"
        p:client-override-xml="file:/objectgrid.xml"
        p:catalog-service-endpoints="${catalogServiceUrl}" />

  ...

</beans>
```

Example client override XML which enables a nearCache (see the Near Cache section for more details):

```xml
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
  xmlns="http://ibm.com/ws/objectgrid/config">
  <objectGrids>
    <objectGrid name="CACHE_REMOTE" txTimeout="60">
      <!-- NOEXP caches' nearCaches use LRU to limit number of nearCache entries per map -->
      <backingMap name="CACHE_NOEXP_.*" template="true"
                  lockStrategy="NONE" ttlEvictorType="NONE" timeToLive="0"
                  copyMode="COPY_TO_BYTES"
                  nearCacheEnabled="true" nearCacheCopyMode="NO_COPY"
                  pluginCollectionRef="LRUevictorPlugins" />
      <!-- EXP caches' nearCaches implicitly use backingMap TTL evictor settings -->
      <backingMap name="CACHE_EXP_.*" template="true"
                  lockStrategy="NONE" ttlEvictorType="LAST_UPDATE_TIME"
                  timeToLive="120" copyMode="COPY_TO_BYTES"
                  nearCacheEnabled="true" />
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```
When a client override XML is successfully loaded, messages such as the following will be printed:

```
[2/10/16 23:50:03:190 EST] 00000000 ObjectGridMan I  CWOBJ2433I: Client-side ObjectGrid settings are going to be overridden for domain DefaultDomain using the URL file:/override-objectgrid.xml.
[2/10/16 23:50:03:758 EST] 00000000 ObjectGridImp I  CWOBJ1128I: The client cache is enabled for maps [IBM_SPRING_PARTITIONED_.*] on the SPRING_REMOTE ObjectGrid.
```

In the above example, the maps using the first template have the LRU evictor specified at the bottom of the XML. The maps using the second template do not specify a pluginCollectionRef but they will implicitly use the TTL evictor because the backingMap specifies a TTL evictor type and time.

The WXS Spring provider enables a "fast fail" mechanism by default. This mechanism exists to allow an application to not hang if a temporary network brownout occurs. Without fastfail, if network connectivity is lost between the client and the WXS server, each request will time out before returning. Fastfail quickly identifies that the network is down and allows all cache requests to return null immediately and reconnect once network connectivity has been restored. This is accomplished with one WXSSpringFastFail[${MAP_NAME}] thread created per map (and if maps are used in different applications with the default classloading policy, one per classloader. This fast fail function may be disabled with -Dcom.ibm.websphere.objectgrid.spring.disable.fastfail=true, in which case TargetNotAvailableExceptions and related exceptions will print FFDCs and a null value will be returned from the cache.

**Monitoring**

There are many ways to monitor WXS:


**Performance Tracing**

See below for additional tracing specific to XIO.

For the overall transaction, use the diagnostic trace com.ibm.ws.objectgrid.SessionImpl=all and calculate the time between the "begin" entry and "commit " exit trace points. That's the lifetime of the transaction on the client. We don't necessarily go to the server immediately after begin() so it's possible if you did the same thing on both the client and the server for the same transaction, you'd get different...
numbers.

On the client side instrumenting `com.ibm.ws.objectgrid.client.RemoteCacheLoader.get()` will give you information on the client side for how long a client get operation is taking.

On the container side instrumenting `com.ibm.ws.objectgrid.ServerCoreEventProcessor.getFromMap()` will give you information on the server side for how long we take to get a value on the server side.

### Offload Caching

WXS is frequently used for HTTP Session persistence instead of a database or Dynacache: [ftp://ftp.software.ibm.com/software/iea/content/com.ibm.iea.wxs/wxs/7.0/Administration/Labs/XS70_HTTPSession_Lab.pdf](ftp://ftp.software.ibm.com/software/iea/content/com.ibm.iea.wxs/wxs/7.0/Administration/Labs/XS70_HTTPSession_Lab.pdf). Keep in mind that the Extreme Scale JVMs will also need to be tuned.

### eXtreme IO (XIO)


### eXtreme Memory (XM)

WebSphere eXtreme Scale v8.6 provides the ability to store cache data outside of the Java heap space. This feature is termed Extreme Memory or XM. Using XM requires the eXtreme IO feature (XIO) introduced in v8.6.


### Data Serialization

**COPY_TO_BYTES**

To optimize serialization with any of these options, you can use the COPY_TO_BYTES mode to improve performance up to 70 percent. With COPY_TO_BYTES mode, the data is serialized when transactions commit, which means that serialization happens only one time. The serialized data is sent unchanged from the client to the server or from the server to replicated server. By using the COPY_TO_BYTES mode, you can reduce the memory footprint that a large graph of objects can use. ([http://www.ibm.com/support/knowledgecenter/en/SSTVLU_8.6.1/com.ibm.websphere.extremescale.doc/cxsserializer.html](http://www.ibm.com/support/knowledgecenter/en/SSTVLU_8.6.1/com.ibm.websphere.extremescale.doc/cxsserializer.html))

### ORB

eXtreme Data Format (XDF)

WebSphere eXtreme Scale v8.6 introduced eXtreme Data Format (XDF) which allows sharing between Java and .NET applications, additional indexing options, automatic versioning, and partitioning through annotations. XDF is the default serialization mode when XIO is enabled and copy mode is COPY_TO_BYTES:


XDF supports serialization of Java objects which do not implement the Serializable interface.

XDF does not compress entries, so data placed in the cache may be larger than other serialization modes and may increase the overhead of network transportation.

CAP Theorem

- Consistency – all clients see the same view, even in the presence of updates
- High Availability – all clients can find some replica of the data, even in the presence of failures
- Partition Tolerance – the system properties are held even when the system is partitioned.

CAP theorem states that a grid can only have two of the three. In WXS version prior to WXS v7.1, grids provide CP services. That is to say that the grid provided consistency (only one place to write the data – the primary shard), and partition tolerance (the grid is capable of providing service even if parts of the grid are network partitioned and unavailable). As of WXS v7.1 we can now have AP grids (Availability and Partition Tolerance).

Queries

WXS provides its own SQL-like query language:


Setting eXtreme Scale tuning options

For standalone client JVM

Create the file objectGridClient.properties in the server's root directory, and add a JVM parameter:

-Dobjectgrid.client.props=objectGridClient.properties

For standalone container JVM:

Create the file objectGridServer.properties and add the JVM command line argument:

-serverProps objectGridServer.properties
**xscmd**

xscmd is the fully supported replacement for the older xsadmin. General:

- Help: xscmd -help
- List available commands: xscmd -lc

The key thing to specify to xscmd is -cep which specifies the list of catalog service endpoints. For example:

```
$ ./xscmd.sh -c listObjectGridNames -cep localhost:4809
...
Grid Name
-------
Grid
```

When the catalog service is running inside WebSphere Application Server (by default, in the deployment manager), and XIO is enabled, the -cep port is the XIO_ADDRESS port.

**Suspend and Resume Status**

The `suspendStatus` command displays the suspend and resume status (ignore the heartbeat option as it only applies to WXS stand alone):

```
$ xscmd.sh -c suspendStatus
...
*** Printing the results of the balance status command for all data grids.

<table>
<thead>
<tr>
<th>Type</th>
<th>ObjectGrid name</th>
<th>Map Set Name</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>placement</td>
<td>Grid</td>
<td>mapSet</td>
<td>Resumed</td>
<td></td>
</tr>
</tbody>
</table>

*** Printing the results of the transport communication failure detection status command for DefaultDomain catalog service domain. The type requested was failoverAll.

<table>
<thead>
<tr>
<th>Type</th>
<th>Domain name</th>
<th>Status</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>failoverAll</td>
<td>DefaultDomain</td>
<td>Resumed</td>
<td></td>
</tr>
</tbody>
</table>
```

When you suspend or resume, the primary catalog logs will contain:

- Placement: CWOBJ1237 for both suspend and resume request attempt, CWOBJ1214 for both suspend and resume when it completes successfully ... the logs will differ with the word "suspend" or "resume" accordingly.
- FailoverAll: CWOBJ1262 for the suspend and resume request attempt, CWOBJ1260 for both suspend and resume when it completes successfully ... the logs will differ with the word "suspend" or "resume" accordingly.
Application Considerations

FIFO Queue

WXS maps may be used as a FIFO queue with the getNextKey method:

Transactions

Twophase transactions will ensure that all changes made to all maps in the transactions are either rolled back or committed:

You can have two maps involved in a transaction without using the Twophase logic. If the two maps are in the same partition, everything will commit or rollback as part of the transaction. WXS will not partially commit a change by having only one map commit and then not doing the other map due to an error; it is always going to be an atomic operation even with a Onephase transaction.

Transaction Callbacks

The TransactionCallback interface may be used to execute code before a Session.commit completes:

WebSphere MQ

WMQ versus WAS SIB

As of Sep. 2011, performance of WebSphere MQ persistent messages is approximately twice as fast as SIBus persistent messages. There is little difference for non-persistent messages.

WebSphere MQ supports clustering of queue managers for enhanced throughput and scalability of administration. There are many examples of production clusters containing thousands of queue managers. WebSphere MQ clustering is extremely flexible, supporting selective parallelism of cluster queues, enabling you to independently tailor the number of instances of each cluster queue. SIBus messaging engines can be clustered within a WebSphere Application Server cluster for throughput and administrative scalability. However, a WebSphere Application Server cluster has a much lower scalability limit than a WebSphere MQ cluster, and if a queue is assigned to a WebSphere Application Server cluster bus member, it is partitioned across all messaging engines in the cluster -- you cannot selectively locate partitions.
WAS Considerations

Listener ports are "stabilized" (no more investment from IBM) and activation specifications are the recommended approach to integrate with WMQ.

Consider various queue properties:

For z/OS, consider this tuning:

If using listener ports, monitor the session pool size:


• Max Connections. Practice is to set this to 2147483647 (maximum possible). This must be set at the same scope as the activation specification. Since activation specifications are generally set at Node scope, Max Connections should be set at Node scope too.
• Connection Concurrency. Practice is to have this property equal 1. Note for WebSphere 8.5, the connectionConcurrency property has been set to 1 as default and made a no-op, so it is not required to explicitly set it. For WebSphere versions earlier than 8.5, this should be set at cell scope.

WAS MQ Resource Adapter

• Versions: http://www-01.ibm.com/support/docview.wss?uid=swg21248089

Best Practices


• Message size and length can affect the performance of the application that processes the message, and the network time of data transmission. Send only essential data in the message.
• Use persistent messages for critical or essential data only. Persistent messages are logged to disk and can reduce the performance of your application.
• Retrieving messages from a queue by message or correlation identifiers will reduce application performance. It causes the queue manager to search all messages in the queue until it finds the desired message. If applications have high-performance requirements, applications should be designed to process messages sequentially.
• The MaxMsgLength parameter stores the value for the maximum size of a message allowed on the queue. The 4 MB default can be changed to better align with your application processing needs, which will have the benefit of using system resources in the most efficient manner.
• Ensure that messaging applications are designed to work in parallel with each other and with multiple instances of applications. The queue manager executes one service request within a queue at a given time to maintain integrity. Avoid programs that use numerous MQPUT calls in
a sync point without committing them. Affected queues can fill up with messages that are currently inaccessible while other applications or tasks might be waiting to get these messages.

- When applications have intermittent message transmission needs, use the MQPUT1 call to put only one message on the queue. For higher volume applications, where multiple messages are being put, consider an alternative to the traditional usage of an MQOPEN call followed by a series of MQPUT calls and an MQCLOSE call.

- Keep connections and queues open if you are going to reuse them instead of repeatedly opening and closing, connecting and disconnecting.

- The maximum number of threads an application can run on a system can affect the performance of the solution, especially on Windows.

- Configure channels with a disconnect interval so that they can go inactive when there is no activity on the channel after a period of time. This will reduce overhead and help improve overall performance.

- MQ performance is commonly bound by disk I/O writes. Ensure that the storage team is involved with disk layouts to ensure the fastest reliable disk writes possible.

- When using clusters: "Adding more than two full repositories often degrades overall performance, because the cluster will need to send additional traffic and spend more time maintaining all of the repositories... [I]t is usually better to create one queue manager with 100 queues as opposed to 100 queue managers with one queue apiece."

Large message depths on your WebSphere MQ queues could cause performance issues. Storing thousands of messages on a single queue is not a best practice. (https://www.ibm.com/developerworks/community/blogs/aimsupport/entry/large_messages_depths_on_mq_queues)

### WMQ Versions

Recent versions of MQ have introduced significant enhancements:

**MQ 7.0:** Major improvements to JMS and publish / subscribe functionality; shared conversations.

**MQ 7.0.1:** Multi-instance queue managers (a standby queue manager can provide failover for an active queue manager; requires a shared network filesystem)

MQ 7.0 and 7.0.1 are both scheduled to go out of support on September 30, 2015.

**MQ 7.1:** Channel-authorization security feature; multiple installations of MQ on a single machine (locations of MQ files and directories are no longer hard-coded / fixed; new setmqinst and dspmqinst commands)

**MQ 7.5:** Integrated Advanced Message Security (AMS) and Managed File Transfer (MFT)

IBM WebSphere MQ V7.1 and V7.5 Features and Enhancements:


**MQ 8.0:** More hostname support; more flexibility when dealing with SSL certificates; ID / password validation by queue managers. Separate Program and Data directories (layout of MQ objects in the filesystem are reorganized)
MQ Documentation

The WebSphere MQ library has links to documentation for all versions of MQ:

Basic MQ Display Commands

- dspmqinst lists the MQ installations on the machine
- dspmqver shows the MQ version and patch level
- dspmq lists queue managers on the local machine, and the status of each one

Performance differences across MQ versions

Official WebSphere MQ Performance Reports are available via the MQ SupportPac site here.

MQ version 8

Performance reports for MQ version 8 are now available via the above link for most platforms. A summary extract from MP6T, the MQ v8 AIX performance report:

IBM MQ V8.0 for AIX has improved performance over V7.5 in almost every area.

- For 2KB non-persistent messages, V8.0 is up to 63% better than V7.5.
- For 2KB persistent messages, V8.0 is up to 16% better than V7.5.
- For 20KB non-persistent messages, V8.0 is up to 11% better than V7.5.
- For 20KB persistent messages, V8.0 is up to 9% better than V7.5.

Specific areas of optimisation in V8.0 include:

- Improved queue manager scaling through lock optimisations in logging and internal catalog code
- Improved TCP buffer defaults.
- Improved client binding performance using SHARECNV(1)
- Reduced client connection time.

MQ version 7.5

Message throughput increases in MQ 7.5:

- 40% on AIX, 80% on Linux, 100% on Windows for persistent messages
- 30% on AIX, and between 40-100% on Linux for non-persistent messages (2KB)
- 500% performance improvement for multi-subscriber message delivery

MQ version 7.1

SupportPac MP6Q for AIX shows the following results in MQ 7.1:
• When testing 2 Kb non-persistent messages in local, client, and distributed queuing environments, Version 7.1 has 28% higher throughput than V6.0.2.11, 30% higher throughput than V7.0, and 35% higher than V7.0.1.6
• When testing 2 Kb persistent messages in local, client, and distributed queuing environments, Version 7.1 has 64% higher throughput than V6.0.2.11, 36% higher throughput than V7.0, and 48% higher throughput than V7.0.1.6

Benchmarks show performance improvements on almost all tests run, when comparing WebSphere MQ V7.1 to V6.0 and V7.0.

Windows and UNIX Performance Tuning

Configuring and tuning WebSphere MQ for performance on Windows and UNIX:

• Queue manager log configuration (applicable only when using persistent messages)
• More tips related to persistent messages:
  • "When processing persistent messages it is recommended to run many instances of the application concurrently in order to optimise the efficiency of the queue manager log."
  • "When processing persistent messages in an application you should ensure that all MQPUT and MQGET activity takes place within a unit of work, or syncpoint as it is sometime referred to, for efficiency purposes."
• Fastpath channels, fastpath listeners
• Queue buffer sizes

WMQ JMS Client


If possible:

• Use non-persistent messages
• Use bindings mode
• Use correlation ID when using selectors
• Use the async put feature
• Use read-ahead when you can, with non-persistent messages
• Use conversation sharing / multiplexed sockets
• Use non-transacted sessions

Resources


1. MP01: WebSphere MQ - Tuning Queue limits
2. MP06: WebSphere MQ with JMS: Get the Best from WMQ and MB Pub/Sub Processing
3. MP7A: WebSphere MQ for Windows V5.3 - Performance tuning for large clusters

**WebSphere DataPower**

Performance tuning links:


**WebSphere Commerce**

**Tuning**


In general, WebSphere Commerce tuning is similar to WAS infrastructure tuning because WebSphere Commerce is a web application using Struts and DB2/Oracle as databases.

**Deployment**

Unless deploying WebSphere Commerce EAR files less than 500 MB in size to production systems, EAR deployment timeout tuning is highly recommended.

This involves several JVM custom properties, especially if you use rollout updates:

- com.ibm.websphere.management.application.updatesync.appExpansionTimeout
- com.ibm.websphere.management.application.updateClusterTask.serverStopWaitTimeout
- com.ibm.websphere.application.updateapponcluster.waitforappsave
- com.ibm.ws.webcontainer.ServletDestroyWaitTime

**Operating system**

It is important to follow the general guidelines set out in the WebSphere Application Server Knowledge Center: [http://www-01.ibm.com/support/knowledgecenter/SSAW57_7.0.0/com.ibm.websphere.installation.nd.doc/info/ae/aetins_prepare.html?cp=SSAW57_7.0.0%2F3-5-1-4&lang=en](http://www-01.ibm.com/support/knowledgecenter/SSAW57_7.0.0/com.ibm.websphere.installation.nd.doc/info/ae/aetins_prepare.html?cp=SSAW57_7.0.0%2F3-5-1-4&lang=en)
Caching

General suggestions
It is recommended to use the NOT_SHARED sharing mode in Dynacache: "For WebSphere Commerce, the recommendation is to use Not shared." (http://www.ibm.com/support/knowledgecenter/en/SSZLC2_8.0.0/com.ibm.commerce.install.doc/tasks/tighorzcluster.htm). For more information, see the HTTP section in the WAS chapter.

REST services caching

Database connection
It is recommended to apply fix pack 8 to unify the DB2 and Oracle database connection: http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/cdealldbconnect.htm?lang=en

"WebSphere Commerce utilities and Ant tasks are updated to reference a single class to configure the acquisition of a database connection. This unified database connection method ensures that each utility and Ant task can reference this class to acquire a database connection, regardless of the JDBC driver that a database uses.

By applying WebSphere Commerce Fix Pack 8 or later, you automatically update the following WebSphere Commerce utilities and processes to use this unified method. You do not need to complete any further action before these utilities and processes begin to use the alldbconnector class to help acquire a database connection."

Feature pack 7 is required to be able to validate database connections: http://www-01.ibm.com/support/knowledgecenter/SSZLC2_7.0.0/com.ibm.commerce.admin.doc/concepts/cdbconnectvalidation.htm?lang=en

"The database connection validation utility tests the connection to the WebSphere Commerce database and logs the connection test results. Use this information to troubleshoot database-related problems. The utility tests whether a connection can be established with the configured parameters of a WebSphere Commerce utility or process. The utility logs the validation results within either the log file of the utility that calls the connection validation utility to run, or within the log file for the validation utility when the utility is run as a stand-alone process."

Redbooks

• Mastering DynaCache in WebSphere Commerce: http://www.redbooks.ibm.com/abstracts/sg247393.html?Open
Troubleshooting

Web server
Tips for using the IHS Access Log feature for WebSphere Commerce sites:

Commerce Server
Use the TrapIt application to monitor log files and fire specific events: http://www-01.ibm.com/support/docview.wss?uid=swg21644180

WebSphere Portal
Portal tuning script:

WebSphere ESB
Processing Large Objects
Ensuring optimum performance is attained on systems processing large objects is an issue commonly faced by users of middle-ware software. In general, objects of 1M or more can be considered to be 'large' and require special attention, please review the following articles for awareness of considerations and tuning / application design advice:

Large Messages dW article: https://www.ibm.com/developerworks/library/ws-largemessaging/
Claim Check Pattern:

Aggregation Design Patterns
There are several application design considerations that should be understood when developing Mediation Flows utilising aggregation design patterns in order to attain optimal performance and avoid unnecessary processing costs. The following article details these design considerations:

Aggregation dW article:
Depending on whether there is a FanIn Mediation Primitive downstream of a FanOut Mediation
Primitive alters the logic within the FanOut mediation Primitive. When using a FanOut Mediation Primitive without an associated FanIn Mediation Primitive an array of SMOs is created up-front before the output terminal is first fired. If there is an associated FanIn then each SMO is created as required instead of all ahead of time. If the SMO is large in size or a large number need to be created (for example, iterating on a large array of elements, firing a large number of times, or a large number of branches), then this can have a significant effect on memory overhead. For example, if you have a 1MB input message and you use a FanOut to iterate over an element that repeats 1000 times, transforms the message and passes on to a JMS queue (without a FanIn), then before the first output terminal fire on the FanOut, 1000 SMOs will be created each of ~1MB in size which would mean you would have a 1GB array allocated to the JVM Heap. You need to be aware of this behaviour when creating your application and tuning the size of the JVM Heap and application threadpools.

**Asynchronous Invocation of Synchronous Services Design Patterns**

In general, synchronous service invocations are recommended, because they have less processing overhead and provide better performance. In some cases however, asynchronous invocations can reduce the overall response time of the application and are preferred, such as in the simultaneous invocation of multiple long-running services. When invoking a synchronous service asynchronously, however, additional processing is incurred in the messaging layer of the product that needs to be understood and tuned appropriately. The following article details these considerations and processing logic:

*Parallel Invocation of Synchronous Services dW article:*

NB. You do not need to switch to asynchronous invocations for JAX-WS Import Bindings to impose specific time-out settings - this can be done by applying an appropriate HTTP Policy configurable through the Administration Console (and can be exported to be brought into the development environment) which does not incur the overhead in the messaging layer of the product - as with the switch to asynchronous invocations. There are some small caveats that need to be considered / understood - please see the "Defining a Timeout on Synchronous JAX-WS Imports" section below.

**Shared Libraries**

The default setting for libraries is share by copy – this means that each Mediation Module referencing a particular library retains its own copy, which can result in bloated and redundant memory usage. You may need to consider shared libraries as detailed in the following technote:


Shared libraries can also benefit run-time performance through reduced serialisation in addition to reducing overall memory footprint - for instance in Lazy Parsing applications employing SCA Binding componentisation.

**Parsing Modes**

Don't mix parsing modes within a deployment. Moving between a Lazy Parsing module an Eager Parsing configured module through SCA Bindings causes increased overhead in processing costs that
should be avoided. Some scenarios will perform better in Eager Parsing mode (lightweight scenarios with small payloads), however, mediation modules which are more complex, or are processing larger payload workloads will typically benefit from Lazy Parsing and can exhibit significant performance improvements (dependant on application design).

**Memory Analyzer Plugin**

IBM Extensions for Memory Analyzer for WebSphere ESB is an extension for IBM Monitoring and Diagnostic Tools for Java™ -- Memory Analyzer, augmenting data structures and providing reports specific to WebSphere ESB. It significantly improves the effectiveness and efficiency of problem diagnosis and resolution, and provides a deeper understanding of your WebSphere ESB deployment. The following article and WebSphere Technical Exchange show you how to use the IBM Extensions for Memory Analyzer for WebSphere ESB to analyze operating system level dumps or portable heap dumps from a WebSphere Enterprise Service Bus solution:

IBM Extensions for Memory Analyzer for WebSphere ESB:

WebSphere Technical Exchange on Memory Analyzer Plugin and APARs associated with memory management:

**Comparative Transformation Technologies (XSLT vs. Business Object Map)**

XSLT Mediation Primitives are designed for applications that have .XSL currently or that want to utilise specific XSLT function.

Business Object Mapper Primitives are designed for improved performance, but may require specific function to be coded manually within the map.

Business Object Maps have some reduced function out of the box, but much can be implemented in simple custom Java utilising the BO API. They provide improved performance, especially for larger message payloads, as they work at the API so do not need to be passed through a transformation engine which will produce bytes causing additional SMO construction and serialization / de-serialization costs. In IBM Integration Designer V8.0 a new mediation primitive has been introduced to enable the developer to switch between the targeted run-time transformation technologies through a simple combo-box - previously a complete re-write of the transformation would be required within the new primitive if a customer wanted to switch between technologies for improved performance.

**First Messing Response Time**

The performance, specifically regarding response time, of the first message entering the system is often of high importance. Typically there is a trade off between artefact initialisation costs being associated with server start-up or first message processing. There are several techniques and product features that can be utilised to improve and control the system when response times are critical for first message processing.

**Synthetic Messages**

First message response times can be improved by priming the Mediation Module with a synthetic message:

Synthetic messages TechNote: http://www-01.ibm.com/support/docview.wss?
This may require the Mediation Module / Components to have an additional "no op" operation or flow path to process the synthetic message without affecting downstream systems, but will result in the vast majority of initialisation costs to have been met prior to the first "production" message entering the system.

**XPath and XSL Pre-compilation**

Pre-compilation of XSL and XPath was introduced in V7.5, these artefacts are now compiled at deploy time rather than on first message for Lazy Parsing Modules (the transformation engine usitilised for Eager Parsing does not have such a concept). This can provide substantial improvements to first message processing time for Lazy Parsing Mediation Flows. The improvement factor is dependent on the number of XSLT Mediation Primitives and XPath statements in the initial path through the Mediation Flow, and the complexity of the XSL / XPath.

**Pre-loading of Mediation Flows**

The option to load Mediation Modules and associated artefacts and resources at server start up, opposed to on first message, was introduced in V7.5.1. A property was exposed that enables the user to use wildcards to select appropriate Mediation Modules and Components and define how many instances of the artefacts to load into the runtime:


Test cases used to evaluate this feature show that >50% improvement can be achieved in initial message processing times - although this is dependent on a number of factors, including the number of components in a project and the complexity of the message definitions. This not only builds the Message Flows but also many of the objects required to model the message structures during server start-up and applies to both Eager and Lazy Parsing Modules.

**Associated APARS**

Several APARs may be required relating to pre-compilation of XSL / XPath and pre-loading of Mediation Flows:
- IC96060: EXTRANEOUS OR MISLEADING ERROR MESSAGES DURING MEDIATION FLOW PRE-LOADING
- IC96845: MULTIPLE PROBLEMS CACHING XSL MAPS RESULTING IN SLOW RESPONSE TIMES AND UPDATES NOT BEING PICKED UP AFTER MODULE RESTART
- IC95917: CACHE PRECOMPILED STYLESHEETS PER CONTEXT CLASSLOADER (http://www-01.ibm.com/support/docview.wss?uid=swg1IC95917)
- IC96799: NULLPOINTEREXCEPTION DURING SERVER STARTUP WHEN PRELOAD VARIABLE IS SET (http://www-01.ibm.com/support/docview.wss?uid=swg1IC96799)
- IC91519: POOR PERFORMANCE/ EXCESSIVE MEMORY USE OF BO MAPPINGS IN BPEL PROCESSES, OR WHEN MAPPINGS ARE APPLIED IN CUSTOM CODE (http://www-01.ibm.com/support/docview.wss?uid=swg1IC91519)
Restricting the Instances of Mediation Flows on the JVM Heap

For WebSpere ESB a Mediation Flow object is required for each concurrent thread executing a unique flow / operation in a Mediation Module. Due to product changes introduced in V7.0 the concepts differ depending on the version of the run-time and the version of the development environment used to generate the run-time artefacts.

V6 Run-time / Applications

EAR files generated prior to V7.0 utilise the EJB Container, whether they are deployed to a V6 or V7 run-time. Each Mediation Module “Application” is represented by a stateless session EJB – the number of EJBs created is controlled as follows:

1. Transport threadpool: Controls maximum concurrency in the system (ie. WebContainer threadpool)
2. Application EJB threadpool (default min=50, max=500): Each Application will create up to the maximum defined number of EJBs in a module-specific pool

If the min value for an EJB pool is set lower then we might free up memory as the pool contracts. The following APAR may be required:


V7 Run-time / Applications

With the exception of EAR files generated prior to V7.0 (but deployed to a V7 run-time) the number of Mediation Flows on the JVM Heap is controlled as follows:

1. Transport threadpool: Controls maximum concurrency in the system (ie. WebContainer threadpool)
2. JVM Managed: Weak / Soft references will clean up unused resources

The references that keep the Mediation Flow objects alive on the JVM Heap have been modified in V7 onwards to enable clean-up to occur when the JVM Heap is under stress. The following APARs may be required:

IC94803: ALLOW FOR GARBAGE COLLECTION OF CERTAIN REFERENCES (http://www-01.ibm.com/support/docview.wss?uid=swg1IC94803)

IC82189: ENABLE MEDIATION FLOWS TO BE GCD WHEN HEAP IS UNDER STRESS (http://www-01.ibm.com/support/docview.wss?uid=swg1IC82189)

Throttling Individual Applications

Often it is required to “throttle” individual applications that may be having an adverse effect on the system (to limit memory usage, or CPU consumption for instance). The concepts and methods differ depending on the version of the run-time and the version of the development environment used to generate the run-time artefacts.

You can throttle applications by tuning / restricting the appropriate threadpools on which they run. This typically has a global impact as many applications may be running on the same threadpool, however, it is possible to isolate applications (or groups of applications) to specific threadpools by creating new
transport chains through which to invoke them.

First create a new threadpool, in the administrative console click “Servers > Server Types > WebSphere application servers > server_name > Thread pools”, then click “New” and fill in the required details. Next create a new transport chain as detailed in the following article:


The next step is to configure the transport chain that has just been created. In the administrative console navigate to the newly created transport chain, click TCP inbound channel and modify the Thread Pool setting to use your new threadpool.

NB. If you create a new web container transport chain, the initial value for the writeBufferSize attribute is 8192, which is too small for most web container transport chains. Navigate to the newly create transport chain, click Web container inbound channel, and specify 32768 (or appropriate value) in the Write buffer size field.

You may also need to configure the appropriate virtual host as described in the following article:


For your application to run on the new threadpool you must invoke it through the appropriate transport chain by using the port you specified on its creation.

V6 Generated Applications

EAR files generated prior to V7.0 utilise the EJB Container, whether they are deployed to a V6 or V7 run-time. Every EJB in WebSphere Application Server has an associated pool with default min, max value of 50,100.

These can be overridden globally or modified on a per Application basis by specifying a system property as follows:

-Dcom.ibm.websphere.ejbcontainer.poolSize=<J2EE-bean-name>=min,max

The J2EE-bean-name is formed by concatenating the application name (without the file extension), the module name (without the file extension) and the name of the bean (“ejb-name” as defined in the bean's deployment descriptor), using a # separator. For example, if you have an application called SMAApp.ear that includes module PerfModule.jar, and module PerfModule.jar uses a bean named TunerBean, the J2EE name of the bean is specified as SMAApp#PerfModule#TunerBean.

If the property is set correctly you should see a line similar to the following output in the system log on first invocation of an operation in the module:

[24/05/11 15:28:02:444 BST] 00000025 EJBMDOrchestr I CNTR0060I: (Min,Max) pool size is (5,100) for bean com.ibm.wsspi.sibx.mediation.flow.ejb.MediationFlowBean

Unfortunately, every WESB module will output a message with the same class name but the pool values will apply to individual beans.

For verification, a trace string of com.ibm.ejs.container.BeanMetaData=all will output the details of
every bean on first invocation including the correct J2EE name needed above and the current pool settings for the EJB.

Reducing the min value of an EJB pool will mean that during quiet spells for a particular application (Mediation Module) the pool will be shrunk down to that minimum value and any associated mediation flow objects will be eligible for GC. The EJB pool is shrunk back down (in increments) to the minimum size after the pool has been inactive for a certain period of time. This can be configured from the admin console at “Application servers > server1 > EJB container”, the setting is labelled "Inactive pool cleanup interval" and defaults to 30 seconds.

Defining a Timeout on Synchronous JAX-WS Imports

Synchronous JAX-WS Bindings do not offer a simple setting to modify the default time-out options, as is available with the asynchronous invocation. However, switching to asynchronous invocations introduces unnecessary overhead that can affect application performance. If you need to set binding / application specific time-out values for a synchronous JAX-WS invocation then this can be achieved by applying an appropriate policy set, and does not incur any additional overheads. To achieve this follow these steps:

1. Create a new policy set in the Administrative Console. Click Services > Policy Sets > Application policy sets > New
2. Add an HTTP Transport policy and configure the time-out values appropriately
3. Save and export the policy set from the Administrative Console
4. Import the policy set into the development environment (Import > WebServices > WebSphere Policy Sets)
5. Attach the policy set to the Import Binding (Properties > Binding > Policy Sets > Default policy set)

It should be noted that if a time-out occurs the exception propagated back to WebSphere ESB is not a modelled fault, thus the failure message is propagated to the fail terminal (the timeout terminal is just for handling time-outs for asynchronous invocations). The SMO failinfo section will appear as follows:

```xml
<failInfo>
  <failureString>javax.xml.ws.WebServiceException: java.net.SocketTimeoutException: Async operation timed out</failureString>
  <origin>External Service</origin>
</failInfo>
```

The reference to "Async operation timed out" just refers to the fact that it is using the Java Async IO API, nothing to do with the existing asynchronous SCA model.

Best Practices and Tuning Red Papers


**Authentication**

**Lightweight Directory Access Protocol (LDAP)**

With WAS com.ibm.ws.security.*=all diagnostic trace, search for "LdapRegistryI > search Entry" to see if a JVM is making LDAP calls

Consider preFetchData to speed up some LDAP operations: [http://www-01.ibm.com/support/docview.wss?uid=swg1PI09171](http://www-01.ibm.com/support/docview.wss?uid=swg1PI09171)

Recent versions of WAS include a basic LDAP search under AdminTask.ldapSearch: [http://www-01.ibm.com/support/docview.wss?uid=swg1PI47190](http://www-01.ibm.com/support/docview.wss?uid=swg1PI47190)

**LdapQuery.ear**


**IBM Tivoli Directory Server (TDS)**


Use cn=monitor to get a snapshot of activity:

```
$ ldapsearch -h ldap_host -s base -b cn=monitor objectClass=* 
```

Key items:

- **currentconnections**: The number of active connections.

  Shows how many established TCP sockets are connected to LDAP; however, WAS has a cache for LDAP connections, so generally this number may not change even if there are a lot of operations over the connections.

- **opscompleted**: The number of completed requests since the server was started.

  Cumulative, so you can take multiple snapshots, and take the difference to find the number of LDAP transactions completed in that time period

- **current_workqueue_size**: The current depth of the work queue.

  The workqueue size is zero if there are no threads waiting for an available worker thread. If all workers are busy, the operations wait in the work queue. Should always be zero; otherwise, there is some contention, either in the LDAP box itself (e.g. CPU), or in the number of threads configured for LDAP, or too much load coming in.

- **available_workers**: The number of worker threads available for work.
if it's at 0 for a long period of time, that's a sign of a busy server, and will usually result in "hangs" from the perspective of the client-side.

The idsmonitor.ksh script can be used to monitor TDS, which includes the cn=monitor output along with time stamps and other information and can be run on an interval: http://www-01.ibm.com/support/docview.wss?uid=swg21282708

Here's a Linux command that converts the idsmonitor.out files to CSV for spreadsheets (and then just adds a column that calculates the difference between rows):

- **opscompleted**: grep -B 15 ^opscompleted idsmonitor.out.20131010.txt | grep -e ^opscompleted -e Date | awk '{printf "%s", substr($0, 7);getline;printf ",%s", substr($0, 15);printf "\n"}'
- **currentconnections**: grep -B 9 ^currentconnections idsmonitor.out.20131010.txt | grep -e ^currentconnections -e Date | awk '{printf "%s", substr($0, 7);getline;printf ",%s", substr($0, 21);printf "\n"}'
- **current_workqueue_size**: grep -B 101 ^current_workqueue_size idsmonitor.out.20131010.txt | grep -e ^current_workqueue_size -e Date | awk '{printf "%s", substr($0, 7);getline;printf ",%s", substr($0, 25);printf "\n"}''

OpenLDAP

See the OpenLDAP chapter in the appendix for non-performance related background.

Monitoring

See available monitors with:

```bash
$ ldapsearch -LLL -W -D cn=Manager,dc=example,dc=com -b cn=monitor objectclass=* 
```

Competition and Migration

Comparing Products

Here are some things to compare when two products are performing differently. Look at the configuration, but also gather evidence on each one (e.g. tracing) to actually confirm or deny whether the feature is in use and the relative cost.

1. Compare "underlying" configurations (at least at a high level) such as the operating system (e.g. CPU, RAM usage, etc.), Java (e.g. maximum heap size, garbage collection overhead, -D parameters, etc.), etc.
2. Security configuration (e.g. authentication provider)
3. Ensure that application logging levels and volume are the same. For example, in one case the default classloading policy of a competitor product picked up a different logging configuration file causing less logging to occur versus WAS.
4. If a different temporary directory is used between products (-Djava.io.tmpdir), make sure this will not have any impact (e.g. if it's on a slower file system). For example, Tomcat changes the
default temporary directory.

5. If the time of a product component (e.g. web service call) is in question, there may be no easy way to compare, so instead consider asking the application to write a log entry at the start and end of each call.

6. If there is a small difference, try to magnify the difference (for example, adding more concurrent users) and then gather data.

7. Use a monitoring product such as ITCAM that works on both products.

8. If you know some aspects of the competition, such as the maximum heap size, then you can test with this same value. If, for example, garbage collection overhead is too high with the same heap size, and there are no other application differences, this may be a sign that some fundamental configuration such as thread pool sizes, data source caching, etc. may be leading to a difference in heap usage and may be the fundamental cause of the difference in performance.

9. Profile your application using tools such as the IBM Java Health Center or more simply by taking multiple thread dumps.

Latest WAS competitive performance numbers:

WAS Migration Performance Differences

If a customer reports that performance is worse after migrating WAS versions, consider the following ideas. In some ways, comparing two versions of the same product (e.g. migration) can also be treated as a "competition" between those two versions using the tips in the previous section.

1. See the general comparison checklist above.

2. What changed? Often times, the hardware or application has changed and this could affect the difference. If possible, try installing both versions and applications in the same operating system instance for comparison.

3. If the migration is from WAS < 8 to WAS >= 8, and on a platform that runs IBM Java and -Xgcpolicy is not specified on WAS >= 8, and -Xgcpolicy was not specified on the previous version or a non-gencon policy was specified, then the default gcpolicy changed to gencon with WAS V8.0. With gencon, part of the young generation (-Xmm, which defaults to 25% of -Xmx) is unavailable for the application (amount changes dynamically based on the tilt ratio), so there would be relatively less Java heap than previously which can cause performance changes.

4. If the migration is from IBM Java < 6.0.1 to IBM Java >= 6.0.1, consider some of the default optimization changes: http://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.60_26/vm626/J9/VM/optimizations_pd.html?lang=en

5. Compare the configurations between versions, first checking the basics such as generic JVM arguments, thread pool configurations, and then more thoroughly. Tools such as the IBM Visual Configuration Explorer tool may be used to compare configurations; however, note that using this tool across major product versions will show many differences in the product that are likely unrelated.

6. WAS 8.5 Traditional includes WVE enabled by default, which includes additional PMI activity amongst other things (ODC rebuilds in the DMGR, etc.), which some customers may notice particularly during idle periods compared to previous versions. WVE may also introduce additionally memory overhead, particularly as the size of the cell increases.
7. Java EE5 modules introduced annotation scanning which can increase startup time and decrease application performance (see the Annotation Scanning section in the WAS chapter).


9. If the migration is from WAS < 8 to WAS >= 8, and the application uses Spring, calls to ApplicationContext.getBean() on beans using the @Async annotation causes higher CPU utilization: [http://www-01.ibm.com/support/docview.wss?uid=swg21648523](http://www-01.ibm.com/support/docview.wss?uid=swg21648523)

10. On z/OS, ensure that WLM service classes and other classifications are the same.

## Recipes

These recipes provide step-by-step instructions to gather and analyze data for the most common types of performance issues. The recipes are organized by topic in a similar way to the rest of the cookbook.

### Contents

- General Recipes
- Operating System Recipes
- Java Recipes
- WAS Traditional Recipes
- WAS Liberty Recipes
- Web Server Recipes
- Caching Recipes

## General Performance Recipes

1. Performance tuning is usually about focusing on a few key variables. We will highlight the most common tuning knobs that can often improve the speed of the average application by 200% or more relative to the default configuration. The first step, however, should be to use and be guided by the tools and methodologies. Gather data, analyze it and create hypotheses: then test your hypotheses. Rinse and repeat. As Donald Knuth says: "Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified. It is often a mistake to make a priori judgments about what parts of a program are really critical, since the universal experience of programmers who have been using measurement tools has been that their intuitive guesses fail." (Donald Knuth, Structured Programming with go to Statements, Stanford University, 1974, Association for Computing Machinery)

2. There is a seemingly daunting number of tuning knobs. We try to document everything in detail in case you hit a problem in that area; however, unless you are trying to squeeze out every last drop of performance, we do not recommend a close study of every point.

3. In general, we advocate a bottom-up approach. For example, with a typical WebSphere
Application Server application, start with the operating system, then Java, then WAS, then the application, etc. Ideally, investigate these at the same time. The main goal of a performance tuning exercise is to iteratively determine the bottleneck restricting response times and throughput. For example, investigate operating system CPU and memory usage, followed by Java garbage collection usage and/or thread dumps/sampling profilers, followed by WAS PMI, etc. See the Major Tools chapter for recommendations on tools to use in each case.

4. One of the most difficult aspects of performance tuning is understanding whether or not the architecture of the system, or even the test itself, is valid and/or optimal.

5. Meticulously describe and track the problem, each test and its results.

6. Use basic statistics (minimums, maximums, averages, medians, and standard deviations) instead of spot observations.

7. When benchmarking, use a repeatable test that accurately models production behavior, and avoid short term benchmarks which may not have time to warm up.

8. Take the time to automate as much as possible: not just the testing itself, but also data gathering and analysis. This will help you iterate and test more hypotheses.

9. Make sure you are using the latest version of every product because there are often performance or tooling improvements available.

10. When researching problems, you can either analyze or isolate them. Analyzing means taking particular symptoms and generating hypotheses on how to change those symptoms. Isolating means eliminating issues singly until you've discovered important facts. In general, we have found through experience that analysis is preferable to isolation.

11. Review the full end-to-end architecture. Certain internal or external products, devices, content delivery networks, etc. may artificially limit throughput (e.g. Denial of Service protection), periodically mark services down (e.g. network load balancers, WAS plugin, etc.), or become saturated themselves (e.g. CPU on load balancers, etc.).

For details, see the first chapter.

Operating System Recipes

- Linux Recipes
- AIX Recipes
- z/OS Recipes
- IBM i Recipes
- Windows Recipes
- Solaris Recipes
- HP-UX Recipes

Linux Recipes

1. **CPU core(s)** should not be consistently saturated. Use tools such as vmstat, top, atop, nmon, perf, SystemTap, etc.

2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk. Use tools such as free, vmstat, /proc/meminfo, top, atop, nmon, etc.

3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times. Use tools such as df, stat, iostat, netstat, ping, nfsiostat, etc.

4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic
effects on performance. Tune TCP/IP socket buffers such as net.core.*mem* and net.ipv4.tcp_*mem* and monitor `netstat -s` for TCP retransmissions.

5. Set `vm.swappiness=0` on systems running Java-based workloads which have light disk file I/O.
6. Consider disabling swap, setting `vm.panic_on_oom=1`, and configuring kernel vmcore dumps with process-level virtual address space information to avoid swap thrashing situations and reduce downtime, whilst analyzing post-mortem vmcores for excessive memory usage, leaks, or undersizing.

7. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis. Use tools such as atop.
8. Review operating system logs for any errors, warnings, or high volumes of messages. Review logs such as `/var/log/messages`, `/var/log/syslog`, etc.
9. Review snapshots of process activity, and for the largest users of resources, review per thread activity. Use tools such as `top -H`.
10. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically. Review CPU steal time in tools such as `vmstat`, `top`, etc.
12. Linux on IBM Power CPUs:
   1. Test with the IBM Java parameter `-Xnodfpbd`
   2. Test with hardware prefetching disabled
   3. Test with idle power saver disabled
   4. Test with adaptive frequency boost enabled
   5. Test with dynamic power saver (favor performance) mode enabled
   6. Use 64-bit DMA adapter slots for network adapters
13. Linux on IBM System z CPUs:
   1. Use QUICKDSP for production guests

For details, see the Linux chapter.

**AIX Recipes**

1. **CPU core(s)** should not be consistently saturated.
2. Generally, physical memory should never be saturated and the operating system should not page memory out to disk.
3. Input/Output interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. TCP/IP and network tuning, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.
9. Bind your processes properly based on system topology.
10. Use MCM memory affinity where appropriate.
11. Find the optimal SMT configuration for the machine.
12. Find the optimal hardware prefetching setting for your workload.
13. Apply recommended tuning for Java applications.
14. For large multi-threaded apps, use profiling to make sure that work is allocated equally amongst threads.
15. For apps that use a lot of network I/O, tune networking parameters.
16. For apps that make heavy use of native memory, experiment with and use the optimal malloc algorithm.
17. Use profiling to evaluate the effects of tuning other parameters.

For details, see the AIX chapter.

**General AIX Performance Problem**

1. Start nmon for essentially unlimited collection with a 60 second interval:
   
   ```
   # su
   # cd /var/tmp/
   # nohup nmon -fT -s 60 -c 1000000 &
   # cat nohup.out
   ```

   Executing this command will start the nmon collector in the background, so explicitly putting it into the background (using `&`) is not necessary. This will create a file with the name $HOST$_$STARTDAY$_$STARTTIME.nmon

   Note that any errors starting nmon (such as inadequate file permissions when trying to write to the specified directory) will go to nohup.out, so it is important to check nohup.out to make sure it started correctly. You can also run `ps -elfx | grep nmon` to make sure it started.

2. Reproduce the problem.
3. When you want to stop nmon, run:

   ```
   # su
   # ps -elf | grep nmon | grep -v grep | awk '{print $4}' | xargs kill -USR2
   ```

4. Gather /var/tmp/*.nmon

**WAS on AIX Performance Problem**

2. Edit aixperf.sh then change SCRIPTSPAN to greater than or equal to the duration of the test. Also ensure JAVA_CORE_INTERVAL is changed so that SCRIPTSPAN is evenly divisible by it. For example:
   
   ```
   SCRIPTSPAN=43200
   JAVA_CORE_INTERVAL=60
   ```
3. Find the process IDs (PIDs) of each application server: `ps -elf | grep java`
4. Start aixperf.sh as root:

   ```
   # su
   # cd /var/tmp/
   ```
5. Reproduce the problem
6. The aixperf.sh script will gather a tprof sample just once when it first starts. Also gather a tprof sample at the peak of the problem:

   # su
   # cd /var/tmp/
   # LDR_CNTRL=MAXDATA=0x80000000 tprof -Rskex sleep 60 >> tprof.out 2>&1

7. Once the reproduction is complete:
8. If aixperf.sh hasn't completed, you can manually stop it:
   1. ps -elf | grep `ps -elf | grep aixperf | grep -v grep` | awk '{print $4}' | xargs kill
   2. Gather any javacore*txt files produced in each WAS JVM's profile directory.
9. Gather /var/tmp/aixperf/* and /var/tmp/tprof/*

**z/OS Recipes**

1. **CPU core(s)** should not be consistently saturated.
2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.
9. Use the Workload Activity Report to review performance.

For details, see the z/OS and WAS Traditional on z/OS chapters.

**IBM i Recipes**

1. **CPU core(s)** should not be consistently saturated.
2. Generally, **physical memory** should never be saturated and the operating system should not page memory out to disk.
3. **Input/Output** interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. **TCP/IP and network tuning**, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.

For details, see the IBM i chapter.

### Windows Recipes

1. **CPU core(s)** should not be consistently saturated.
2. Generally, physical memory should never be saturated and the operating system should not page memory out to disk.
3. Input/Output interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. TCP/IP and network tuning, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.
9. Use Perfmon to review performance activity.
10. Use the Windows Performance Toolkit to review sampled native processor usage.

For details, see the Windows chapter.

### Solaris Recipes

1. **CPU core(s)** should not be consistently saturated.
2. Program memory should not page out of RAM.
3. Input/Output interfaces such as network cards and disks should not be saturated, and should not have poor response times.
4. TCP/IP and network tuning, whilst sometimes complicated to investigate, may have dramatic effects on performance.
5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.
6. Review operating system logs for any errors, warnings, or high volumes of messages.
7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.
8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.

For details, see the Solaris chapter.

### HP-UX Recipes

1. **CPU core(s)** should not be consistently saturated.
2. Generally, physical memory should never be saturated and the operating system should not page memory out to disk.

3. Input/Output interfaces such as network cards and disks should not be saturated, and should not have poor response times.

4. TCP/IP and network tuning, whilst sometimes complicated to investigate, may have dramatic effects on performance.

5. Operating system level statistics and optionally process level statistics should be periodically monitored and saved for historical analysis.

6. Review operating system logs for any errors, warnings, or high volumes of messages.

7. Review snapshots of process activity, and for the largest users of resources, review per thread activity.

8. If the operating system is running in a virtualized guest, review the configuration and whether or not resource allotments are changing dynamically.

For details, see the HP-UX chapter.

**Java Recipes**

1. Tune the maximum Java heap size (-Xmx):
   1. Enable verbose garbage collection (-verbose:gc) which prints statistics on garbage collection to files and generally has an overhead less than 1%. Use a tool such as the IBM Garbage Collection and Memory Visualizer to analyze the verbosegc output. The proportion of time spent in garbage collection versus application processing time should generally be less than 10% and ideally less than 1%.
   2. Garbage collection will adapt heap size to keep occupancy between 40% and 70%. Heap occupancy over 70% causes frequent GC cycles... Heap occupancy below 40% means infrequent GC cycles, but cycles longer than they needs to be... The maximum heap size setting should therefore be 43% larger than the maximum occupancy of the application.

2. Consider the particular type of garbage collector to use (see the comparison table in either the IBM Java or Oracle/HotSpot Java chapters).

3. Ensure there is no memory leak with long running tests.

4. If using a generational collector such as IBM gencon/balanced or the Oracle JVM:
   1. Ensure tests run through full/tenured collections and ensure those pause times are not too long.
   2. Ensure that there is a sawtooth pattern in the heap usage after collection. Otherwise, the heap size may be too small or the nursery too big.
   3. Generally, the sawtooth should drop about 25% of the heap size on full collections.

5. Total pause times over a few seconds should be routinely investigated.

6. Use a profiler such as IBM Java Health Center or Java Mission Control with a particular focus on the profiling and lock contention analysis. Otherwise, use periodic thread dumps to review JVM activity with the IBM Thread and Monitor Dump Analyzer tool.

7. Object allocation failures for objects greater than 5MB should generally be investigated.

8. Take a system dump or HPROF heapdump during peak activity and review it with the IBM Memory Analyzer Tool to see if there are any areas in the heap for optimization.

9. Review the stderr and stdout logs for any errors, warnings, or high volumes of messages (e.g. OutOfMemoryErrors).

10. If running multiple JVMs on the same machine, consider pinning JVMs to sets of processor...
cores and tuning -Xgcthreads/-XcompilationThreads or -XX:ParallelGCThreads.

11. In general, if memory usage is very flat and consistent, it may be optimal to fix -Xms = -Xmx. For widely varying heap usage, -Xms < -Xmx is generally recommended. You may get the best of both worlds by setting -Xms to the lowest steady state memory usage, -Xmx/f1.0 to eliminate shrinkage, -Xminf to avoid compaction before expansion, and -Xmine to reduce expansions.

12. Request a thread dump and search its output for "deadlock" to ensure that no threads are deadlocked (thus reducing throughput).

13. If using the IBM Java Runtime Environment:
   1. In most cases, the gcenon garbage collection policy works best, with the key tuning being the maximum heap size (-Xmx) and maximum nursery size (-Xmn).
   2. Test with large pages (-Xlp). These are enabled by default in recent versions, but may require operating system configuration for full enablement.
   3. Test with -Xaggressive.
   4. Test with -XtlhPrefetch.
   5. Consider enabling IBM Java Health Center (-Xhealthcenter) by default so that you can attach to particular processes if they start to have trouble.
   6. If using IBM Java >= 7 SR3, the IBM JCE security provider and recent Intel, AMD, or POWER >= 8 CPUs, then AESNI hardware acceleration for AES encryption and decryption can be exploited with -Dcom.ibm.crypto.provider.doAESInHardware=true. This can reduce TLS overhead by up to 35%.
   7. If the static IP address of the node on which Java is running is unlikely to change, use -Dcom.ibm.cacheLocalHost=true to reduce localhost lookup time.
   8. Take a javacore thread dump and review the Java arguments (UserArgs) and Environment Variables sections for uncommon or debug options.
   9. If physical memory allows, increase the size of the shared class cache (-Xshareclasses).

14. If using the Oracle Java Runtime Environment:
   1. In most cases, the -XX:+UseParallelOldGC garbage collection policy works best, with the key tuning being the maximum heap size (-Xmx) and maximum new generation size (-XX:MaxNewSize).
   2. Set -XX:+HeapDumpOnOutOfMemoryError.
   3. When using ergonomics, consider tuning -XX:MaxGCPauseMillis and -XX:GCTimeRatio.
   4. When fine-tuning is required, consider disabling ergonomics (-XX:+AdaptiveSizePolicy) and tune the SurvivorRatio (-XX:SurvivorRatio).

For details, see the Java chapter and the chapter for your particular JVM.

Java Profilers Recipe

1. In most cases, sampling profilers are used first and tracing profilers are only used for fine grained tuning or deep dive analysis.
2. Analyze any methods that use more than 1% of the reported time in themselves.
3. Analyze any methods that use more than 10% of the reported time in themselves and their children.
4. Analyze any locks that have large contention rates, particularly those with long average hold times.
WAS Traditional Recipes

1. Upgrade to the latest version and fixpack of WAS as it has a history of making performance improvements over time.
2. Use the Performance Monitoring Infrastructure (PMI) to monitor various statistics such as thread pool usage, database connection pools, etc. Use a tool such as IBM ITCAM, IBM Tivoli Performance Viewer (TPV) in the Administrative Console, the WAS Performance Tuning Toolkit, etc.
3. Thread pools should not be consistently saturated.
4. Database connection pools should not be consistently saturated.
5. Monitor response times of key application components (e.g. servlets, databases, MDBs, etc.).
6. Apply the production performance tuning template.
7. On WAS 8.5, explicitly install and switch to the most recent version of Java available using manageSdK.
8. Switch to High Performance Extensible Logging (HPEL) and disable JMX log notifications (-Dcom.ibm.ejs.ras.disablerasnotifications=true).
9. Review SystemOut/SystemErr/HPEL, FFDC and application logs for any errors, warnings, or high volumes of messages.
10. If possible, configure and use servlet caching/Dynacache.
11. Don't neglect to monitor and tune the node agents and deployment manager (particularly garbage collection).
12. Ensure that when WAS fixpacks have been applied, the correct service release of Java was also upgraded: https://www-304.ibm.com/support/docview.wss?uid=swg27005002
13. If Single Sign On (SSO) is enabled, test whether performance is better with web inbound attribute propagation enabled (default) or disabled.

For details, see the WAS Traditional chapter.

General WAS Traditional Performance Problem

1. Make sure the logs are capturing as much as possible:
   1. Administrative Console > Troubleshooting > Logs and Trace > server name > JVM Logs (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/utrb_jvmlogs.html). These can also be changed dynamically on the Runtime tab.
   2. Maximum size = 100MB
      Maximum Number of Historical Log Files = 5
2. Ensure verbose garbage collection is enabled: http://www-01.ibm.com/support/docview.wss?uid=swg21114927. On certain operating systems and WAS versions, you may enable verbosegc dynamically at runtime. Otherwise, you will need to restart to apply the change. See the Java chapters for details.
3. Ensure that PMI is enabled either with the "Basic" level (this is the default) or with a "Custom" level (see WAS chapter on which counters are recommended):
   1. Enable PMI logging to files, either with a monitoring product such as ITCAM or with the built-in TPV logger (https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd
1. **Important note**: all of these steps must be done after every application server restart. This can be automated with wsadmin:
   
   https://raw.githubusercontent.com/kgibm/problemdetermination/master/scripts/was/tpvlogging.py

2. Login to the Administrative Console and go to: Monitoring and Tuning > Performance Viewer > View Logs
3. Select all relevant application servers and click "Start Monitoring"
4. Click each application server
5. Click on server > Settings > Log
6. Duration = 300000
   - Maximum File Size = 50
   - Maximum Number of Historical Files = 5
   - Log Output Format = XML
7. Click Apply
8. Click server > Summary Reports > Servlets
9. Click "Start Logging"

4. For IBM Java, enable IBM Health Center in headless mode:
   
   1. Update to the latest Health Center agent in the WAS java directory:
      
   
   2. Choose one of these methods to start Health Center (http://www-01.ibm.com/support/docview.wss?uid=swg21657760):
      
      1. Start it dynamically: `${WebSphere}/java/bin/java -jar ${WebSphere}/java/jre/lib/ ext/healthcenter.jar ID=${PID}
         -Dcom.ibm.java.diagnostics.healthcenter.data.collection.level=headless
         -Dcom.ibm.java.diagnostics.healthcenter.headless.files.max.size=104857600
         -Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=10
      2. Restart the JVM adding the following generic JVM arguments:
         -Xhealthcenter:level=headless
         -Dcom.ibm.java.diagnostics.healthcenter.headless.files.max.size=104857600
         -Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=10

5. If there is a web server in front of WAS, see the Web Server recipes.
6. Archive and truncate any existing logs for each server in (`{WAS}/profiles/{PROFILE}/logs/${SERVER}`) and also archive and remove the FFDC logs (`{WAS}/profiles/{PROFILE}/ffdc/*`).
7. Reproduce the problem.
8. Gather the Performance, Hang, or High CPU issue MustGather for your operating system:
   
9. Gather periodic thread dumps (see the WAIT tool in Java - Profilers). This is accomplished through the Performance MustGathers above.
10. After the problem has been reproduced, gracefully stop the application servers (this is needed to produce Health Center logs).
11. Gather:
   1. Server logs under $\{\text{WAS}/\text{profiles}/\text{PROFILE}/\text{logs}/\text{SERVER}\}/: \text{SystemOut*.log}, \text{SystemErr*.log}, \text{native_stderr.log}, \text{native_stdout.log}
   2. FFDC logs under $\{\text{WAS}/\text{profiles}/\text{PROFILE}/\text{logs/ffdc/}\*
   3. Javacores, heapdumps, and system dumps: $\{\text{WAS}/\text{profiles}/\text{PROFILE}/javacore* \}
      $\{\text{WAS}/\text{profiles}/\text{PROFILE}/heapdump* \}
      $\{\text{WAS}/\text{profiles}/\text{PROFILE}/core*
   4. PMI logs: $\{\text{WAS}/\text{profiles}/\text{PROFILE}/\text{logs/tpv/}\*
   5. Health Center logs: $\{\text{WAS}/\text{profiles}/\text{PROFILE}/\text{logs/}\text{tpv/}\text{.hcd}
   6. server.xml for each server: $\{\text{WAS}/\text{profiles}/\text{PROFILE}/\text{config/cells/}\}
      $\{\text{CELL}/nodes/\text{NODE}/servers/\text{SERVER}/\text{server.xml}
   7. The output of the Performance MustGather

12. Review all WAS logs for any errors, warnings, etc. (see WAS - Basics).

13. Review IHS messages in access_log, error_log, and plugin log to see if requests are coming in and if there are errors (check response codes). Also review mpmstats to see what the threads are doing.


15. Review thread dumps
   1. Review patterns (e.g. WAIT tool) and check for deadlocks and monitor contention (e.g. TMDA tool).

16. Review operating system data for WAS and IHS nodes
   1. If CPU time is high, review if it's user or system.
      1. Review per-process and per-thread CPU data for details.
   2. Check virtualization steal time
   3. Check run queue length and any blocked threads
   4. Check for hundreds or thousands of swap-ins
      1. If high, check memory statistics such as file cache, free memory, etc.

17. Review PMI data for the key performance indicators such as the WebContainer thread pool ActiveCount, database connection pool usage, servlet response times, etc. (see WAS - PMI). Try to isolate the problem to particular requests, database queries, etc (duration or volume).

18. Review Health Center data
   1. Review hot self and tree methods and monitor contention.

19. If using a database, review the response times. Try to isolate the problem to particular queries (duration or volume). Check for lock contention.

**Large Topologies Recipe**

1. Use clusters to scale horizontally and vertically, and to support failover and easier administration. If using WAS >= 8.5, consider using dynamic clusters.
   - Very large topologies also employ multiple cells for the same application(s). This allows for deployment of new application versions or configurations to only one of the cells; if the change breaks, it affects only that cell. Multiple cells can be problematic if significant database schema changes are made.

2. If using the High Availability Manager or any functions that require it (e.g. EJB WLM, SIB, etc.):
   - Processes such as application servers and node agents must be in the same core group, or part of bridged core groups.
   - In general, the number of processes in a single core group should not exceed 200.
Practically, this number is limited by the CPU usage, heartbeat intervals, and number of available sockets.

- The members of a core group should be on the same LAN.
- The members of a cell should not communicate with one another across firewalls as that provides no meaningful additional security and complicates administration.
- Create dedicated preferred coordinators for a core group with a large default maximum heap size (e.g. -Xmx1g).
- If using core group bridges, create dedicated bridge servers with a large default maximum heap size (e.g. -Xmx1g).
- Start or stop groups of processes at the same time to reduce the effects of view changes.
- Change the HAM protocols to the latest versions: `IBM_CS_WIRE_FORMAT_VERSION` and `IBM_CS_HAM_PROTOCOL_VERSION`.

3. If you are not using the High Availability Manager, it is not recommended to disable it, but instead to create multiple cells or bridged core groups.

For details, see the Scaling and Large Topologies section of the WAS Traditional Profile chapter.

**Request Metrics Recipe**

1. In addition to the General Traditional WAS Performance Problem recipe, enable WAS Request Metrics to standard logs. This will have a significant performance overhead.
2. WebSphere Administrative Console > Monitoring and Tuning > Request Metrics
3. Ensure "Prepare Servers for Request metrics collection" is checked (by default, it is).
4. Under "Components to be instrumented," select "All"
5. Under "Trace level," select "Performance_Debug"
6. Under "Request Metrics Destination," check "Standard Logs"
7. Click "OK," save and synchronize. If "Prepare Servers for Request metrics collection" was already checked (the default), then the application server does not need to be restarted.

**Tune a Thread Pool**

Tuning a thread pool is one of the most important performance exercises. The optimal maximum thread pool size is the point at which throughput is maximized and resource utilizations (such as CPU) are at comfortable levels. The key thing to remember is that you can only conclude anything when observing a thread pool running at its maximum concurrency (i.e. nothing can be concluded if there is a lesser load than that which fills up the thread pool coming in), and when the mix of work is representative of normal user behavior.

1. Start at a maximum thread pool size of X.
2. Observe the system running with X concurrent threads and gather diagnostics such as throughput, response times, processor usage, monitor contention, and any other relevant resource usage.
3. If one of the resources exceeds (or is significantly below) a comfortable utilization level (for example, average CPU more than 90% utilized, or it is only 5%), then perform a binary search [https://en.wikipedia.org/wiki/Binary_search_algorithm] on X.

For example, let's say we start at 50 WebContainer threads and load the system to 50 concurrent threads. Let's say we're focused on CPU and we want it to be no more than 90% in the worst case. We run a test and CPU is 100%, so we take half and go to 25 maximum threads. We run another test and
CPU is still 100%, so we go to 12. With 12, CPU is 50% which is no longer saturated but now it's not utilizing the CPU as much as we'd like, so we increase by half the difference: CEILING(12 + (25-12)/2) = 19. With 19, CPU is 95%, so we subtract half the difference again: CEILING(18 - (18-12)/2) = 15. With 15, CPU is 90% and we're done.

Here is some pseudo code showing the algorithm:

```python
# Target and TargetWindow are in terms of the Measurement, e.g. CPU %
# Minimum, Maximum, and X are in terms of the thread pool size
Target = T
TargetWindow = W
Minimum = N
Maximum = M
Measurement = PerformTest(X)

loop {
    if (Measurement < (T - W)) {
        N = X
        X = CEILING((M - X) / 2)
    } else if (Measurement > (T + W)) {
        M = X
        X = CEILING((X - N) / 2)
    } else {
        Target met. Print X, Measurement
        BreakLoop()
    }

    Measurement = PerformTest(X)
}
```

**HTTP Sessions**

1. Consider reducing the session timeout (default 30 minutes) and average session size to reduce memory and processing pressures.
2. Consider if session failover is required as it increases complexity and decreases performance. The alternative is to affinitize requests and surgically store any critical state into a database.
3. Use session persistence (database) or WebSphere eXtreme Scale over memory-to-memory replication.
4. Consider using timed updates to save session state.

For more information, see the [HTTP section](http://someurl) of the WAS Traditional Profile chapter.

**Security Recipe**

1. Consider disabling Java 2 security if you can guarantee, to a high confidence, that you know what code is being put on the server and who has access.
2. Consider eliminating secure communications on an already secure part of the LAN. For example, if a web server is in the DMZ, the connection to the application servers may be secured, but all other connections behind the DMZ may be unsecured.
3. Monitor the utilization of the authentication cache and increase its size if it's full and there's heap space available. Also consider increasing the cache timeout.
4. Consider changing administrative connectors from SOAP to RMI to utilize persistent connections.
If using LDAP:

1. Select the reuse connection option

For more details, see the Security section of the WAS Traditional Profile chapter.

**Connection Pool Hangs in `createOrWaitForConnection`**

This recipe provides 3 possible strategies for dealing with connection pool hangs.

---

**Strategy 1: Increase connection pool size maximum to 2x+1 (x = thread pool size maximum)**

When an application is using multiple, simultaneous connections in the same thread, ensure the connection pool size is at least one more than the maximum number of threads so that the threads should never run out of available connections in the pool.

If the application opens 3 or more simultaneous connections you may have to experiment and try 3x+1 or 4x+1 as necessary.

**Monitor**

From the command line execute the above command periodically to capture the number of open connections to the database port number on the same node the application server(s) are running on.

```
$ netstat -an | grep ESTABLISHED | grep <port#> | wc -l
```

**Caveat**

This increases the number of overall database connections from each individual application server. Make sure the database is configured and capable of handling the total number of connections for the sum of all JVMs.

---

**Strategy 2: Disable "shareable" connections**

Set `globalConnectionTypeOverride=unshared` to disable shareable connections:

[https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tdat_connpoolman.html](https://www.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/tdat_connpoolman.html) and retest the application. See the Java Database Connectivity (JDBC) page.

**Monitor**

The application and SystemOut.logs to see if any unexpected exceptions or logic errors occur.

From the command line use:

```
tail -f SystemOut.log
```

**Caveat**

This will cause application problems for applications using container managed EJBs. Typically
this strategy works for Web Container applications accessing databases directly through JDBC.

Strategy 3: Fix the application code

The previous two strategies are operational - run time changes to try to deal with an application that uses multiple simultaneous connections in the same thread request execution. The previous two strategies may not be operationally possible due to resource limitations in the environment. In this case the only way to fix the problem is to fix the application code to never have more than one connection open at a time within the same thread request execution.

Monitor

Javacore files to ensure that there are no threads stuck in createOrWaitForConnection.

Caveat

This may require extensive re-design of the application code and can be a time consuming fix.

Threads in socketRead0 in JDBC calls

JDBC calls can sometimes get stuck on socket read calls to the database if some rather nasty network problems exist or if there is a firewall, between the application server and the database, that aggressively closes long-lived connections (some organizations have security reasons to prevent long-lived connections).

The way to determine if network problems exist is to use tcpdump (AIX, Linux) or snoop (Solaris) to capture the packets into files. One can then use Wireshark to read the capture files. If you see issues like "unreassembled packets", "lost segments" or "duplicate ACK" errors then most likely the network is experiencing serious difficulties affecting the server.

WebSphere Application Server is reporting hung threads in the SystemOut.log and a hung thread message as follows:

```java
[1/2/12 1:23:45:678 EDT] 0000001c ThreadMonitor W WSVR0605W: Thread "WebContainer : 15" (00000045) has been active for 722010 milliseconds and may be hung.
There is/are 1 thread(s) in total in the server that may be hung.
  at java.net.SocketInputStream.socketRead0(Native Method)
  at java.net.SocketInputStream.read(SocketInputStream.java:141)
  at com.ibm.db2.jcc.t4.z.b(z.java:199)
  at com.ibm.db2.jcc.am.nn.executeQuery(nn.java:698)...
```

There exists no deadlock or timeout recorded in the logs, even when there are lock timeout (LOCKTIMEOUT) and deadlock check time (DLCHKTIME) settings defined that are greater than 0.
**Strategy 1: Apply socketRead timeouts**

If threads hang on socketRead0 calls that never seem to get a response then the only way to deal with them is by applying timeouts.

For DB2, use this parameter:

`blockingReadConnectionTimeout`

The amount of time in seconds before a connection socket read times out. This property applies only to IBM Data Server Driver for JDBC and SQLJ type 4 connectivity, and affects all requests that are sent to the data source after a connection is successfully established. The default is 0. A value of 0 means that there is no timeout.

For Oracle's database use:

`oracle.jdbc.ReadTimeout`

Set the timeout to a reasonable value. The actual value depends on how long is the longest running transaction for the particular application connected to a specific database. If the longest transaction is, for example, 10 seconds then a reasonable value for the timeout could be 12 seconds.

**Monitor**

Watch the SystemOut.log file and ensure that hung thread messages do not appear again.

**Caveats**

If the timeout is set too low for the longest running transactions then those transactions will fail.

**Slow or Hung Application**

The SystemOut.log contains entries of:

WSVR0605W: Thread `<threadname>` has been active for `<time>` and may be hung. There are `<totalthreads>` in total in the server that may be hung.

**Recommendations**

**Automatically generate thread dumps**

Enable javacore thread dumps to be generated when a hung application has been detected following the instructions in

[Configuring the WebSphere Application Server hung thread detector to automatically produce a javacore or thread dump on a WSVR0605W message](#)

Otherwise it will be impossible to troubleshoot the problem.

**Open a PMR**

Analyzing thread dumps requires a certain level of proficiency with the IBM Support Assistant.
If no one at the organization knows how to analyze the thread dump open a PMR with IBM Support who can provide the data analysis necessary to help pinpoint where the hang occurred.

---

**Strategy 1: Ran out of disk space OR Slow file system I/O OR Anti-Virus Protection OR Active backup**

A thread dump (javacore) shows a lot of threads in a stack that looks like

```
"WebContainer : 89" daemon prio=10 tid=0x01683c58 runnable
 (0x73f7d000..0x73f7faf0)
   at java.io.FileOutputStream.writeBytes(Native Method)
   at java.io.FileOutputStream.write(FileOutputStream.java:260)
   at
   - locked (0x97ff0230) (a com.ibm.ejs.ras.WrappingFileOutputStream)
   at java.io.PrintStream.write(PrintStream.java:412)
```

**Threads in java.io.FileOutputStream.writeBytes**

---

**Strategy 2: JDBC Connection Pool hang**

- [Connection Pool Hangs in createOrWaitForConnection.](#)
- [Threads in socketRead0 in JDBC calls.](#)

---

**Strategy 3: Check trace levels**

It is not unusual for someone to enable trace, then not turn it off.

---

**Strategy 4: Check PMI levels**

It is not unusual for someone to enable all PMI counters which can severely degrade performance. Enable only the PMI metrics necessary to gauge the health of the system.

---

**Threads in java.io.FileOutputStream.writeBytes**

A thread dump (javacore) shows a lot of threads in a stack that looks like

```
"WebContainer : 89" daemon prio=10 tid=0x01683c58 runnable
 (0x73f7d000..0x73f7faf0)
   at java.io FileOutputStream.writeBytes(Native Method)
   at java.io FileOutputStream.write(FileOutputStream.java:260)
```
Strategy 1: Ran out of disk space OR Slow file system I/O OR Anti-Virus Protection OR Active backup

Can be due to either running out of disk space on the file system or the file system I/O is slow (i.e. high latency connection to a SAN).

- Check if the file system is full. If the file system is full then archive and delete unnecessary files.
- If the file system is slow then change the application configuration to point to a more robust file system.
- Anti-Virus protection may be aggressively scanning the file system providing limited access to all other applications to the file system.
- Active backup that is aggressively accessing the file system providing limited access to all other applications to the file system.

Monitor

- If the disk is highly utilized (for example, 80%), notify the appropriate system administrators.
- File system performance. If aggressive disk usage is detected above your threshold, notify the appropriate system administrators.
  - Investigate re-architecting the environment so that not all the applications are pointed to the same file system.
  - If the problem is related to local disk speed replace local disks with faster disks.
  - If this is due to too many vertically deployed application servers consider expanding the infrastructure horizontally.
- If Anti-Virus protection is aggressively accessing the file system then reconfigure the process not to aggressively access the file system.
- If a backup is aggressively accessing the file system then either reconfigure the process not to aggressively access the file system or investigate using other disk replication techniques.

Caveats

May require restarting the application servers which may require an outage.

Some of the recommended re-architecture/infrastructure can be quite extensive and time/labor consuming. Plan appropriately.

Make sure to investigate a multiple cell infrastructure deployment to help improve availability.

WAS Liberty Recipes

1. Upgrade to the latest version and fixpack of WAS as it has a history of making performance
improvements over time.

2. Liberty has a primary thread pool instead of multiple thread pools in the WAS Traditional Profile and this is auto-tuned based on throughput. Set executor maxThreads for manual control.

3. Database connection pools should not be consistently saturated: `<connectionManager maxPoolSize="X" ... />`.

4. Monitor response times of key application components (e.g. servlets, databases, MDBs, etc.).

5. Set the maximum keep alive requests to unlimited: `<httpOptions maxKeepAliveRequests="-1" />`.

6. Review logs for any errors, warnings, or high volumes of messages.

7. Enable requestTiming and eventLogging with reasonable thresholds.

8. Enable the monitor feature and related connector to have the ability to dynamically monitor metrics of a running JVM.

9. In production environments, disable the application auto-update monitoring capabilities.

10. Switch to Binary Logging to improve logging performance.

11. When using messaging, review the maxBatchSize and maxConcurrency of JMS activation specification configurations.

12. Consider enabling the HTTP access log with response times.

13. If there is a performance problem, test with `<execututor coreThreads="X" maxThreads="X" />` where X = NUM_CPUs*2. If this is better, consider opening a support case to investigate why the default is not optimal.

For details, see the WAS Liberty chapter.

**Web Servers Recipes**

1. The maximum concurrency variables (e.g. MaxClients for IHS and optionally/rarely MaxConnections for the WAS plugin) are the key tuning variables. Ensure MaxClients is not saturated through tools such as mpmstats or mod_status, while at the same time ensuring that the backend server resources (e.g. CPU, network) are not saturated (this can be done by scaling up the backend, sizing thread pools to queue, optimizing the backend to be faster, or with plugin MaxConnections).

2. Use WAS Traditional clusters or WAS Liberty collectives to scale out work over multiple systems and processes, both for fault tolerance and increasing capacity.

3. Clusters of web servers are often used with IP sprayers or caching proxies balancing to the web servers. Ensure that such IP sprayers are doing "sticky SSL" balancing so that SSL Session ID reuse percentage is higher.

4. Load should be balanced evenly into the web servers and back out to the application servers. Compare access log hit rates for the former, and use WAS plugin STATS trace to verify the latter.

5. Review snapshots of thread activity to find any bottlenecks. For example, increase the frequency of mpmstats and review the state of the largest number of threads.

6. Review the keep alive timeout. The ideal value is where server resources (e.g. CPU, network) are not saturated, maximum concurrency is not saturated, and the average number of keepalive requests has peaked (use mpmstats or mod_status).

7. Check the access logs for HTTP response codes (e.g. %s for IHS) >= 400.

8. Check the access logs for long response times (e.g. %D for IHS).

9. Review access and error logs for any errors, warnings, or high volumes of messages.
10. Use WAS plugin DEBUG or TRACE logging to dive deeper into unusual requests such as slow requests, requests with errors, etc. Use an automated script for this analysis: [https://github.com/covener/plugin-tools/blob/master/scanplugin.pl](https://github.com/covener/plugin-tools/blob/master/scanplugin.pl)

11. Fine-tuning of SSL ciphers or other MPM configuration directives is unlikely to have a big impact.

For details, see the [Web Servers chapter](#). Also review the [operating systems chapter](#).

**IHS & WAS Plugin Performance**

1. In the `conf/httpd.conf` file, find the section for `mod_mpmstats.c` and change `ReportInterval` to:
   ```
   ReportInterval 30
   ```

2. In the `conf/httpd.conf` file, find the `CustomLog` directive for your `access_log`. By default this is:
   ```
   CustomLog logs/access_log common
   ```

   The last part of that line, in this example "common" is the name of the LogFormat to use. Find this LogFormat. By default this is:
   ```
   LogFormat "%h %l %u %t "\"\r\" %>s %b" common
   ```

   Change this to:
   ```
   LogFormat "%h %l %u %t "\"\r\" %>s %b %{RH}e %{WAS}e %D" common
   ```

3. Archive and truncate existing `access.log`, `error.log`, and `http_plugin.log` files.

4. Save `httpd.conf` and restart the IHS servers (either gracefully or fully).

5. Reproduce the problem.

6. Gather
   1. `access.log`
   2. `error.log`
   3. `http_plugin.log`
   4. `httpd.conf`
   5. `plugin-cfg.xml`

7. Review all logs for any errors, warnings, etc.

8. Review the response times. Try to isolate the problem to particular requests (duration or volume).


10. Review incoming rate and distribution of requests (see Web Servers).

11. Review `http_plugin.log`:

**Some Users Reporting Bad Performance**

This recipe provides a strategy for identifying which JVM a user is on in order to track down performance issues reported by that user.
Strategy 1: Add Logging of JSESSIONID in IHS to Identify the clone-id of the JVM the user is on

Log Format "%h ... JSESSIONID="%{JSESSIONID}C"

The above string (...) is the rest of the Log Format line in IHS. To print out the JSESSIONID cookie in the IHS access log add the above JSESSIONID string to the end of the Log Format directive. This is helpful because the JSESSIONID string contains the clone the user has established their JSESSIONID with. This way if a user is having problems the administrator will know which clone the user was pinned to. This helps immensely with troubleshooting because the administrator knows which log file they need to look at when the error occurs. Test this out in the test environment first. Then in production make sure disk space is monitored to ensure that the disk does not run out of space because of the additional logging data.

Monitor

access.log on the IHS server. Use the clone-id in the JSESSIONID cookie to identify the JVM. Conduct appropriate troubleshooting steps on that JVM to understand the users' performance problems.

Caveat

Whilst the JSESSIONID is only an identifier, the administrators need to ensure that file system security is locked down so that other users on the node do not have access to the IHS logs.

Caching Recipes

The costs and benefits of caching are discussed in the Caching chapter. This recipe is a checklist of caching to review in a typical WAS installation:

1. If available, enable the Java shared class and ahead-of-time compilation caches. WAS enables this by default, but you can increase the size if you have available memory. See the Java chapter.
2. Pre-compile Java Server Pages (JSPs). See the WAS chapter.
3. If possible, utilize the WAS Dynacache feature to cache servlet responses. See the HTTP section in the WAS chapter.
4. The application should set standardized response headers that indicate caching (e.g. Cache-Control in HTTP). See the Applications chapter.
   1. An alternative is to use a web server such as IHS to apply cache headers to responses based on rules. See the Web Servers chapter.
5. If possible, use the WebSphere eXtreme Scale (WXS) product to maximize data caching (see below).
6. Consider using an edge cache such as the WebSphere Caching Proxy. See the Web Servers chapter.
7. If using WebSphere Commerce, set Dynacache caches' sharing modes to NOT_SHARED.
Troubleshooting Recipes

- Troubleshooting Operating System Recipes
- Troubleshooting Java Recipes
- Troubleshooting WAS Traditional Recipes

Troubleshooting Operating System Recipes

- Troubleshooting Linux Recipes
- Troubleshooting AIX Recipes

Troubleshooting Linux Recipes

1. Set unlimited core dump and file ulimits.
2. A well-tuned operating system is a better behaving operating system, so also review the Linux tuning recipes.

Troubleshooting AIX Recipes

1. Set unlimited core dump and file ulimits.
2. A well-tuned operating system is a better behaving operating system, so also review the AIX tuning recipes.

Troubleshooting Java Recipes

1. Troubleshooting IBM Java Recipes

Troubleshooting IBM Java Recipes

1. Write verbosegc to rotating log files; for example, -Xverbosegclog:verbosegc.%Y%m%d.%H%M%S.%pid.log,5,51200
2. On recent versions of IBM Java, enable Health Center to write to rotating log files; for example, -Xhealthcenter:level=headless
   -Dcom.ibm.java.diagnostics.healthcenter.headless.files.max.size=268435456
   -Dcom.ibm.java.diagnostics.healthcenter.headless.files.to.keep=4
3. Periodically monitor stderr (native_stderr.log in WAS Classic, console.log in WAS Liberty) for "JVM" messages, including those noting the production of javacores, heapdumps, core dumps, and snap dumps.
4. Create a dedicated filesystem for JVM artifacts such as javacores, heapdumps, Snaps, and core dumps (so that if it fills up, the program directories are not affected) and use the -Xdump directory option to change the default directory of these artifacts; for example, -Xdump:directory=${SOME_DIRECTORY} and also set -Xdump:nofailover if there is any concern about filling up the temporary directory.
5. Starting with Java 6.26 SR1, a core dump is produced on the first OutOfMemoryError. Assuming core dumps are configured correctly to be untruncated (see the Troubleshooting Operating System Recipes), then the core dump is sufficient to investigate OutOfMemoryErrors.
(a PHD may always be extracted from a core) and you should disable heapdumps with
-Xdump:heap:none

6. Enable large object allocation tracking and monitor stderr for JVMDUMP039I messages; for example, -Xdump:stack:events=allocation,filter=#10m

7. Consider setting the excessive garbage collection threshold (at which point the JVM is considered to be out of Java memory) to something more aggressive; for example,
-Xgc:excessiveGCratio=80

8. A well-tuned JVM is a better-behaving JVM, so also review the Java tuning recipes.


**Java OutOfMemoryError (OOM)**

1. If you have verbosegc enabled (you should), then review the verbosegc log:
   1. Review the allocation failure right before the OOM to see its size. The cause of the OOM may be an abnormally large allocation request.
   2. Review the pattern of heap usage to see if there are signs of a leak.

2. By default, an OutOfMemoryError should produce a javacore.txt file. Review the javacore:
   1. Review the reason code for the OutOfMemoryError at the top of the javacore. For example:
      
      1TISIGNINFO   Dump Event "systhrow" (00040000) Detail
      "java/lang/OutOfMemoryError" "Java heap space" received
   2. Review the maximum heap size in the javacore. In general, if -Xmx is <= 512M, a sizing exercise may not have been done. For example:
      
      2CIUSERARG    -Xmx3800m
   3. Search for the word "deadlock." If you find "Deadlock detected !!!" then investigate the cause of the deadlock. A deadlock often indirectly causes an OutOfMemory because the deadlocked threads and any threads waiting for a monitor owned by the deadlocked threads are hung indefinitely and this may hold a lot of memory on those threads or impede other processing that cleans up memory.
   4. In some cases, the thread that proximately causes the OOM is reported as the "Current thread." Review the stack for anything abnormal. For example:
      
      1XMCURTHDINFO Current thread

3. Review the coredump or heapdump in the Memory Analyzer Tool.

**Troubleshooting Memory Leaks**

1. Analyze verbosegc in GCMV. If there is a positive slope in the plot "Used heap (after global collection)", then there may be a leak.
   1. The default plot of "Used heap (after collection)" for generational collectors may sometimes look like a leak if there hasn't been a global collection recently, thus why it's best to only look at heap usage after global collections.
   2. There are cases where a positive slope after global collections is not a leak such as SoftReference caches: [https://publib.boulder.ibm.com/httpserv/cookbook/Major_Tools-IBM_Memory_Analyzer_Tool.html#Major_Tools-IBM_Memory_Analyzer_Tool_MAT-SoftReferences](https://publib.boulder.ibm.com/httpserv/cookbook/Major_Tools-IBM_Memory_Analyzer_Tool.html#Major_Tools-IBM_Memory_Analyzer_Tool_MAT-SoftReferences)
   3. Consider the magnitude of the heap growth relative to the heap size. Small relative growths may be reasonable. Caches may need to be populated up to some limit before
they stabilize.

2. If there's evidence of a leak, take an OS core dump (IBM Java) or HPROF dump (Oracle Java) and load into IBM MAT. Things to consider:
   1. Review the largest objects (e.g. a leak in some cache):
   3. Review the number (e.g. thread leaks) and retained heaps (e.g. not a leak but simply too much load) of threads: http://help.eclipse.org/mars/index.jsp?topic=%2Forg.eclipse.mat.ui.help%2Fgettingstarted%2Fbasictutorial.html
   4. Run the IBM Extensions for Memory Analyzer Classloader Leak Detection under WAS > ClassLoaders
   5. Perform a general review of the dump (class histogram, top consumers, etc.)

3. If a single core dump is inconclusive, take two or more OS core dumps (IBM Java) or HPROF dumps (Oracle Java) from the same process and compare them in IBM MAT to find the growth(s): https://publib.boulder.ibm.com/httpserv/cookbook/Major_Tools-IBM_Memory_Analyzer_Tool.html#Major_Tools-IBM_Memory_Analyzer_Tool_MAT-Comparing_Heap_Dumps. The more time between dumps the better to make finding the growth(s) easier. Ideally, use a monitoring tool to track heap usage after full GC and take the second dump after a relative growth of > 10%.

4. The most common leaks are:
   1. Large objects (byte arrays, etc.)
   2. Java collections such as Maps and Lists, often a bug removing items or a cache. One technique the tool uses in the leak suspect report, but which can also be run manually under Leak Identification > Big Drops in Dominator Tree, is to find a large difference between the retained heap of an object and its largest retained reference. For example, imagine a HashMap that retains 1GB and the leak is due to a bug removing objects so objects continue to be added to the HashMap. It is common in such a case for every individual object to be small.

5. Proactive:
   1. Use a monitoring tool to track heap usage after full GC and alert if heap usage is above 70% and gather dumps.
   2. If using WAS traditional, Memory Leak and Excessive Memory Usage Health Condition:
   3. If using Java ODR, Configure Memory Overload Protection and put a server into maintenance mode to investigate:
   4. If using WAS traditional, Application ClassLoader Leak Detection:
Troubleshooting WAS Traditional Recipes

1. Periodically monitor WAS logs for warning and error messages.
2. Set the maximum size of JVM logs to 256MB and maximum number of historical files to 4.
3. Set the maximum size of diagnostic trace to 256MB and maximum number of historical files to 4.
4. Change the hung thread detection threshold and interval to something smaller that is reasonable for the application, and enable a limited number of thread dumps when these events occur. For example:
   1. com.ibm.websphere.threadmonitor.threshold=60
   2. com.ibm.websphere.threadmonitor.interval=30
   3. com.ibm.websphere.threadmonitor.dump.java=15
   4. com.ibm.websphere.threadmonitor.dump.java.track=3
5. Enable periodic thread pool statistics logging with the diagnostic trace
   *=info:Runtime.ThreadMonitorHeartbeat=detail
6. Monitor for increases in the `Count` column in the FFDC summary file (${SERVER}_exception.log) for each server, because only the first FFDC will print a warning to the logs.
7. Review relevant timeout values such as JDBC, HTTP, etc.
8. A well-tuned WAS is a better-behaving WAS, so also review the WAS Tradtional tuning recipes.
9. Review the Troubleshooting Operating System Recipes and Troubleshooting Java Recipes.

Troubleshooting WAS Traditional on z/OS

1. Increase the value of server_region_stalled_thread_threshold_percent so that a servant is only abended when a large percentage of threads are taking a long time (see also). Philosophies on this differ, but consider a value of 10.
2. Set control_region_timeout_delay to give some time for work to finish before the servant is abended; for example, 5.
3. Set control_region_timeout_dump_action to gather useful diagnostics when a servant is abended; for example, SVCDUMP
4. Reduce the control_region_${PROTOCOL}_queue_timeout_percent values so that requests time out earlier if they queue for a long time; for example, 10.
5. If necessary, apply granular timeouts to particular requests (see also).
6. Run listTimeoutsV85.py to review and tune timeouts.

Troubleshooting Tips

1. While investigating a problem, try to eliminate or reduce any uncontrolled changes to variables such as configuration or application changes. Introduce changes methodically.
2. Try to find the smallest, reproducible set of steps that causes the problem.
3. If a problem cannot be reproduced in a test environment, consider disallowing real traffic from coming into a particular production node, and then debugging on that node.
**Root Cause Analysis (RCA)**

Root cause analysis is the search for the primary, sufficient condition that causes a problem. The first danger of "root" cause analysis is that you may think you're done when you're not. The word "root" suggests final, but how do you know you're done? For example, in a situation earlier this year, the problem was high user response times. The proximate cause was that the processors were saturated. The processors were being driven so heavily because System.gc was being called frequently, forcing garbage collections. This was thought to be the "root cause" so somebody suggested using the option -Xdisableexplicitgc to make calls to System.gc do nothing. Everyone sighed relief; root cause was found! Not so. The System.gc's were being called due to native OutOfMemoryErrors when trying to load classes (and -Xdisableexplicitgc doesn't affect forced GCs from the JVM handling certain NOOMs). After much more investigation, we arrived at a very complex causal chain in which there wasn't even a single cause:

The second danger of root "cause" analysis is that it suggests a single cause, which obviously isn't always the case.

Properly understood and with all the right caveats, RCA is fine, but it is rarely properly understood and rarely comes with caveats. Once someone declares that "root cause" has been found, most people are satisfied, especially if removing that cause seems to avoid the problem. It is interesting that the term "root" has gained such a strong hold, when it is clearly too strong of a term. It's possible that "root" was added to "cause analysis," because without "root," some people might stop at the first cause, but perversely, the phrase has caused the exact same sloppiness, laziness and false sense of accomplishment that it was probably designed to avoid. However, given that both suffer from the same problem, "root cause analysis" is worse than "cause analysis" because at least the latter is more open ended. Instead, the term "causal chain" is preferred because it seems to define the investigation in terms of a chain of causes and effects and is more suggestive of the open-endedness of this chain.

Some popular troubleshooting patterns are the Apollo methodology and KT (Kepner-Tregoe).
Analysis versus Isolation

A common aspect to a problem is that an application worked and then the environment (WAS, etc.) was upgraded and the application stopped working. Many customers then say, "therefore, the product is the root cause." It is easy to show that this is a logical fallacy (neither necessary nor sufficient) with a real world example: A customer upgraded from WAS 6.1 to WAS 7 without changing the application and it started to throw various exceptions. It turned out that the performance improvements in WAS 7 and Java 6 exposed existing concurrency bugs in the application.

It is not wrong to bring up the fact that a migration occurred. In fact, it's critical that you do. This is important information, and sometimes helps to quickly isolate a problem. However, people often make the argument that the fact of the migration is the key aspect to the problem. This may or may not be true, but what it does do is elevate the technique of isolation above analysis, which is often a time-consuming mistake (http://www.ibm.com/developerworks/websphere/techjournal/0806_supauth/0806_supauth.html).

Analysis is the technique of creating hypotheses based on observed symptoms, such as exceptions, traces, or dumps. In the above example, the customer experienced java.util.ConcurrentModificationExceptions in their application, but they did not analyze why.

Isolation is the technique of looking at the end-to-end system instead of particular symptoms, and simplifying or eliminating components until the problem is isolated, either by the process of elimination, or by finding the right symptoms to analyze. Saying that the migration is the key aspect to the problem is really saying that the first step is to understand what changed in the migration and then use that to isolate which changed component caused the problem. As the above example demonstrates, changes such as performance improvements may have unknown and unpredictable effects, so isolation may not help.

In general, start with analysis instead of isolation. You should certainly bring up any changes that occurred right before the problem (migration, etc.), but be careful where this leads everyone. If analysis leads to a dead end, that's when I start to use isolation, including comparing changes, but even in this case, comparing product versions is difficult; many things change.

Problem Diagnostics Lab Tookit (PDTK)

The Problem Diagnostics Lab Toolkit is an EAR file that can be installed inside WebSphere Application Server and used to simulate various problems such as OutOfMemoryErrors: https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=b1c67c6c-af3e-4d27-b1bd-40caef331ac0

IBM Support

For problems that fall within the scope of your IBM Support Contract (note that some performance issues do not), but cannot be resolved within a reasonable time, we always recommend you open a Problem Management Record (PMR) at the appropriate severity level (ftp://ftp.software.ibm.com/software/server/handbook/webhndbk.pdf). What is reasonable will depend on how important the application is to the business and the Service Level Agreements (SLAs) the application is expected to deliver.
After opening a PMR with IBM Support, we will need data about your specific issue. In order to expedite analysis, WAS provides instructions on the data collection steps for various problem scenarios in a list of MustGathers (http://www-01.ibm.com/support/docview.wss?uid=swg21145599). Once you have collected the relevant data, upload it to the PMR (see below). Once IBM has received the data, we will begin to analyze it and provide a response within the designated time limits depending on the severity level (see the previous handbook link).

If you feel the PMR needs more attention call the local toll free number (http://www.ibm.com/planetwide/) and ask the person who answers the phone to speak with the "duty manager." Provide the duty manager with your PMR number and the specific issue you feel needs to be addressed.

If you are evaluating WAS software and have not purchased licenses, you cannot open a PMR; however, a business partner may be able to open PMRs while working with you through the Business Partner Technical Strategy and Enablement (BPTSE) - Developer Services program (formerly known as WebSphere Competency Center).

How to Upload Data to a Case

HTTP: https://www.secure.ecurep.ibm.com/app/upload_sf

SFTP:
1. Prepend file(s) with the case number. For example, TS001599663_somefile.zip. You may also prepend with Case. For example, Case_TS001599663_somefile.zip
2. sftp anonymous@sftp.ecurep.ibm.com
3. Press Enter at the password prompt
4. cd toibm/websphere
5. put TS001599663_somefile.zip

How to Upload Data to a PMR

1. Create a README.txt file that describes the test and any relevant time periods or problem time stamps.
2. Create a single archive per machine with a descriptive name and the PMR number, replacing commas with periods (e.g. 12345.000.000.test1.zip).
3. Follow upload instructions provided at: http://www-01.ibm.com/support/docview.wss?uid=swg21153852
   1. Option #1: If the files are less than 20MB in total, simply send an email to a PMR just like any other update, with the files attached, to techsupport@ecurep.ibm.com and the subject, "PMR 12345,678,901"
   2. Option #2: Secure browser upload: https://www.ecurep.ibm.com/app/upload
   3. Option #3: FTP to the host ftp.ecurep.ibm.com (ftp ftp.ecurep.ibm.com), login with the user anonymous, and your email address as the password, set transfer mode to binary (bin), change directory to the proper directory (cd toibm/websphere), and upload the correctly named file prefixed with the PMR number (put 12345.000.000.test1.zip).
Escalating Support


Troubleshooting Operating Systems

Additionally, see the chapter for your particular operating system:

- Linux
- AIX
- z/OS
- IBM i
- Windows
- Solaris
- HP-UX

Debug Symbols

Some applications use native libraries (e.g. JNI; .so, .dll, etc.) to perform functions in native code (e.g. C/C++) rather than through Java code. This may involve allocating native memory outside of the Java heap (e.g. malloc, mmap). These libraries have to do their own garbage collection and application errors can cause native memory leaks, which can ultimately cause crashes, paging, etc. These problems are one of the most difficult classes of problems, and they are made even more difficult by the fact that native libraries are often "stripped" of symbol information.

Symbols are artifacts produced by the compiler and linker to describe the mapping between executable code and source code. For example, a library may have a function in the source code named “foo” and in the binary, this function code resides in the address range 0x10000000 - 0x10001000. This function may be executing, in which case the instruction register is in this address range, or if foo calls another function, foo’s return address will be on the call stack. In both cases, a debugger or leak-tracker only has access to raw addresses (e.g. 0x100000a1). If there is nothing to tell it the mapping between foo and the code address ranges, then you'll just get a stack full of numbers, which usually isn't very interesting.

Historically, symbols have been stripped from executables for the following reasons: 1) to reduce the size of libraries, 2) because performance could suffer, and 3) to complicate reverse-engineering efforts. First, it's important to note that all three of these reasons do not apply to privately held symbol files. With most modern compilers, you can produce the symbol files and save them off. If there is a problem, you can download the core dump, find the matching symbols locally, and off you go.

Therefore, the first best practice is to always generate and save off symbols, even if you don't ship them with your binaries. When debugging, you should match the symbol files with the exact build that produced the problem. This also means that you need to save the symbols for every build, including one-off or debug builds that customers may be running, and track these symbols with some unique identifier to map to the running build.

The second best practice is to consider shipping symbol files with your binaries if your requirements allow it. Some answers to the objections above include: 1) although the size of the distribution will be larger, this greatly reduces the time to resolve complex problems, 2) most modern compilers can create
fully optimized code with symbols [A], and 3) reverse engineering requires insider or hacker access to
the binaries and deep product knowledge; also, Java code is just as easy to reverse engineer as native
code with symbols, so this is an aspect of modern programming and debugging. Benefits of shipping
symbols include: 1) not having to store, manage, and query a symbol store or database each time you
need symbols, 2) allow "on site" debugging without having to ship large core dumps, since oftentimes
running a simple back trace or post-processing program on the same machine where the problem
happened, with symbols, can immediately produce the desired information.

As always, your mileage may vary and you should fully test such a change, including a performance
test.

**Eye Catcher**

Eye-catchers are generally used to aid in tracking down native memory leaks or native
OutOfMemoryErrors. After you've checked all the obvious culprits, at some point you may have to
manually page through a hexdump. An eye-catcher, as its name suggests, is some sequence of bytes
that has a low probability of randomly appearing in memory. If you see one of your eye-catchers, it's
possible that you've found one of your allocations.

For example, below is a simple C program which leaks 10 MyStruct instances into the native heap and
then waits indefinitely so that I can grab a coredump of the process:

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#define EYECATCHER_MYSTRUCT 0xDEADFAD0

typedef struct {
    int eyeCatcher;
    int myData;
} MyStruct;

tmain(int argc, char** argv) {
    sigset_t sigmask;
    MyStruct *p;
    int i;

    for (i = 0; i < 10; i++) {
        p = (MyStruct*)malloc(sizeof(MyStruct));
        printf("Alloced struct @ 0x%0X\n", p);
        p->eyeCatcher = EYECATCHER_MYSTRUCT;
        p->myData = 123*i;
    }

    printf("Hello World. Waiting indefinitely...\n");
    sigemptyset(&sigmask);
    sigaddset(&sigmask,SIGCHLD);
    sigsuspend(&sigmask);
}
```

Now, we can find all of these structures in a hexdump. In this example, integers are stored in little
We can see the ten allocations there. Now this works best if the eye catcher is cleared before freeing the allocation:

```c
void destroy(MyStruct *p) {
    p->eyeCatcher = 0;
    free(p);
}
```

Otherwise, you won't know for sure when you see the eye catcher that the memory block is allocated or free.

(Note: the eye catcher just happened to be on a word boundary. It's possible that it spanned multiple lines or across the 8 byte boundary. The best way to search for eye catchers is through some type of automation such as gdb extensions.)

A lot of people prefer strings instead of integers. This solves the problem of big- and little-endianness and it's normally easier to spot these strings:

```c
#define EYECATCHER_MYSTRUCT2 "DEADFAD0"

typedef struct {
    char eyeCatcher[9]; // Add 1 to the length of the eye catcher, because
    // strcpy will copy in the null terminator
    int myData;
} MyStruct2;
...
```

```c
for (i = 0; i < 10; i++) {
    p2 = (MyStruct2*)malloc(sizeof(MyStruct2));
    printf("Alloced struct @ 0x%0X\n", p2);
    strcpy(p2->eyeCatcher, EYECATCHER_MYSTRUCT2);
    ...
```
p2->myData = 123*i;
}

...

$ hexdump -C core.6940 | grep DEADFAD0 | tail -10
00002df0  00 00 00 00 44 45 41 44  46 41 44 30 00 00 00 00
|....DEADFAD0....|
00002e10  00 00 00 00 44 45 41 44  46 41 44 30 7b 00 00 00
|....DEADFAD0{....|
00002e30  00 00 00 00 44 45 41 44  46 41 44 30 f6 00 00 00
|....DEADFAD0....|
00002e50  00 00 00 00 44 45 41 44  46 41 44 30 71 01 00 00
|....DEADFAD0....|
00002e70  00 00 00 00 44 45 41 44  46 41 44 30 ec 01 00 00
|....DEADFAD0{....|
00002e90  00 00 00 00 44 45 41 44  46 41 44 30 67 02 00 00
|....DEADFAD0....|
00002eb0  00 00 00 00 44 45 41 44  46 41 44 30 e2 02 00 00
|....DEADFAD0....|
00002ed0  00 00 00 00 44 45 41 44  46 41 44 30 5d 03 00 00
|....DEADFAD0....|
00002ef0  00 00 00 00 44 45 41 44  46 41 44 30 d8 03 00 00
|....DEADFAD0....|
00002f10  00 00 00 00 44 45 41 44  46 41 44 30 53 04 00 00
|....DEADFAD0S....|

Here are some other considerations:

1. If you're writing native code that is making dynamic allocations, always use eye catchers. Yes, they have a small overhead. It's worth it. The evidence for this recommendation is that most large, native products use them.
2. You can put an "int size" field after the eye catcher which stores the size of the allocation (sizeof(struct)), which makes it easier to quickly tell how much storage your allocations are using.
3. You can wrap all allocations (and deallocations) in common routines so that this is more standard (and foolproof) in your code. This is usually done by having an eye catcher struct and wrapping malloc. In the wrapped malloc, add the sizeof(eyecatcherstruct) to the bytes requested, then put the eye catcher struct at the top of the allocation, and then return a pointer to the first byte after sizeof(eyecatcherstruct) to the user.

**Troubleshooting Linux**

**General Troubleshooting Commands**

- Print system page size: getconf PAGESIZE
- The `ausyscall` command converts a syscall number to the syscall name. Example:

  ```
  $ ausyscall 221
  fadvise64
  ```
The kill command is used to send a signal to a set of processes or to terminate them:

```
$ kill ${PID}
```

Without arguments, the SIGTERM (15) signal is sent. To specify a signal, use the number or name of the signal. For example, to send the equivalent of Ctrl+C to a process, use one of:

```
$ kill -2 ${PID}
$ kill -INT ${PID}
```

To list all available signals:

```
$ kill -l
1) SIGHUP     2) SIGINT     3) SIGQUIT     4) SIGILL     5) SIGTRAP
  6) SIGABRT    7) SIGBUS     8) SIGFPE      9) SIGKILL    10) SIGUSR1
 11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
 16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP...
```

SIGSTOP may be used to completely pause a process so that the operating system does not schedule it. SIGCONT may be used to continue a stopped process.

The pkill command is the same as kill except that it uses other matching terms rather than the process ID. For example, to send SIGKILL to any processes with "server1" in the available command line:

```
$ pkill -KILL -f server1
```

Process core dumps

Core dumps are normally written in the ELF file format. Therefore, use the readelf program to find all of the LOAD sections to review the virtual memory regions that were dumped to the core:

```
$ readelf --program-headers core
```

```
Program Headers:
Type Offset VirtAddr PhysAddr
FileSiz MemSiz Flags Align
NOTE 0x00000000000003f8 0x0000000000000000 0x0000000000000000 1
LOAD 0x0000000000000ca4 0x0000000000000000 0x0000000000000000 1
LOAD 0x0000000000001ca4 0x0000000000000000 0x0000000000000000 1
```

Request core dump (also known as a "system dump" for IBM Java)

Additional methods of requesting system dumps for IBM Java are documented in the Troubleshooting IBM Java and Troubleshooting WAS chapters.

1. The gcore command pauses the process while the core is generated and then the process should continue. Replace ${PID} in the following example with the process ID. You must have permissions to the process (i.e. either run as the owner of the process or as root). The size of the core file will be the size of the virtual size of the process (ps VSZ). If there is sufficient free
space in physical RAM and the filecache, the core file will be written to RAM and then asynchronously written out to the filesystem which can dramatically improve the speed of generating a core and reduce the time the process is paused. In general, core dumps compress very well (often up to 75%) for transfer. Normally, the gcore command is provided as part of the gdb package. In fact, the gcore command is actually a shell script which attaches gdb to the process and runs the gdb gcore command and then detaches.

```bash
# gcore ${PID} core.`date +%Y%m%d.%H%M%S`.dmp
```

There is some evidence that the gcore command in gdb writes less information than the kernel would write in the case of a crash (this probably has to do with the two implementations being different code bases).

2. The process may be crashed using `kill -6 ${PID}` or `kill -11 ${PID}` which will usually produce a core dump.

IBM proposed a kernel API to create a core dump but it was rejected for security reasons and it was proposed to do it in user space.

### Core dumps from crashes

When a crash occurs, the kernel may create a core dump of the process. How much is written is controlled by coredump_filter:

Since kernel 2.6.23, the Linux-specific /proc/PID/coredump_filter file can be used to control which memory segments are written to the core dump file in the event that a core dump is performed for the process with the corresponding process ID. The value in the file is a bit mask of memory mapping types (see mmap(2)). ([http://man7.org/linux/man-pages/man5/core.5.html](http://man7.org/linux/man-pages/man5/core.5.html))

When a process is dumped, all anonymous memory is written to a core file as long as the size of the core file isn't limited. But sometimes we don't want to dump some memory segments, for example, huge shared memory. Conversely, sometimes we want to save file-backed memory segments into a core file, not only the individual files. /proc/PID/coredump_filter allows you to customize which memory segments will be dumped when the PID process is dumped. coredump_filter is a bitmask of memory types. If a bit of the bitmask is set, memory segments of the corresponding memory type are dumped, otherwise they are not dumped. The following 7 memory types are supported:

- (bit 0) anonymous private memory
- (bit 1) anonymous shared memory
- (bit 2) file-backed private memory
- (bit 3) file-backed shared memory
- (bit 4) ELF header pages in file-backed private memory areas (it is effective only if the bit 2 is cleared)
- (bit 5) hugetlb private memory
- (bit 6) hugetlb shared memory

Note that MMIO pages such as frame buffer are never dumped and vDSO pages are always
dumped regardless of the bitmask status. When a new process is created, the process inherits the bitmask status from its parent. It is useful to set up coredump_filter before the program runs.

For example:

```bash
$ echo 0x7 > /proc/self/coredump_filter
$ ./some_program
```

https://www.kernel.org/doc/Documentation/filesystems/proc.txt

**Process Virtual Address Space**

The total virtual and resident address space sizes of a process may be queried with ps:

```bash
$ ps -o pid,vsz,rss -p 14062
PID  VSZ    RSS
14062 44648 42508
```

Details of the virtual address space of a process may be queried with


```bash
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 fd:02 22151273 /bin/cat...
```

This will produce a line of output for each virtual memory area (VMA):

```bash
$ cat /proc/self/maps
00400000-0040b000 r-xp 00000000 fd:02 22151273 /bin/cat...
```

The first column is the address range of the VMA. The second column is the set of permissions (read, write, execute, private). The third column is the offset if the VMA is a file, device, etc. The fourth column is the device (major:minor) if the VMA is a file, device, etc. The fifth column is the inode if the VMA is a file, device, etc. The final column is the pathname if the VMA is a file, etc.

The sum of these address ranges will equal the `ps VSZ` number.

In recent versions of Linux, smaps is a superset of maps and additionally includes details for each VMA:

```bash
$ cat /proc/self/smaps
00400000-0040b000 r-xp 00000000 fd:02 22151273 /bin/cat
Size:                 44 kB
Rss:                  20 kB
Pss:                  12 kB...
```

The Rss and Pss values are particularly interesting, showing how much of the VMA is resident in memory (some pages may be shared with other processes) and the proportional set size of a shared VMA where the size is divided by the number of processes sharing it, respectively.

The pmap command prints similar information to smaps although it does not print PSS:

```bash
$ pmap -x 7638
Address Kbytes RSS Dirty Mode Mapping
```
smaps

The total virtual size of the process (VSZ):

\[
\text{\$ grep } ^\text{Size} \text{ smaps } | \text{ awk } '{\text{print } \$2}' | \text{ paste -sd+ | bc | sed 's/$/\times1024/' | bc}
\]

3597316096

The total resident size of the process (RSS):

\[
\text{\$ grep Rss smaps } | \text{ awk } '{\text{print } \$2}' | \text{ paste -sd+ | bc | sed 's/$/\times1024/' | bc}
\]

897622016

The total proportional resident set size of the process (PSS):

\[
\text{\$ grep Pss smaps } | \text{ awk } '{\text{print } \$2}' | \text{ paste -sd+ | bc | sed 's/$/\times1024/' | bc}
\]

891611136

In general, PSS is used for sizing physical memory.

Print sum VMA sizes greater than 60MB:

\[
\text{\$ grep -v -E } ^\text{\[a-zA-Z_\]+:} \text{ smaps } | \text{ awk } '{\text{print } \$1}' | \text{ sed 's/-/\,/' | perl -F, -lane 'print hex($F[1])-hex($F[0]);' | sort -n | grep } ^\text{\[6\].....}$ | \text{ paste -sd+ | bc}
\]

840073216

Sort VMAs by RSS:

\[
\text{\$ cat smaps | while read line; do read line2; read line3; read line4; read line5; read lin6; read line7; read line8; read line9; read line10; read line11; read line12; read line13; read line14; echo $line, $line3; done | awk '{\text{print } $(NF-1)\times1024, \$1}' | sort -n}
\]

gdb

Loading a core dump

A core dump is loaded by passing the paths to the executable and the core dump to gdb:

\[
\text{\$ gdb } \{\text{PATH_TO_EXECUTABLE}\} \{\text{PATH_TO_CORE}\}
\]

To load matching symbols from particular paths (e.g. if the core is from another machine):

1. Run gdb without any parameters
2. (gdb) set solib-absolute-prefix ./
3. (gdb) set solib-search-path .
4. (gdb) file ./path_to_java
5. (gdb) core-file ./path_to_core

Batch execute some gdb comments:
$ gdb --batch --quiet -ex "thread apply all bt" -ex "quit" $EXE $CORE

**Common Commands**

- Print current thread stack: bt
- Print thread stacks: thread apply all bt
- List all threads: info threads
- Switch to a different thread: thread N
- Print register: p $rax
- Print current instruction: x/i $pc
- Disassemble function at address: disas 0xff
- Print structure definition: ptype struct malloc_state
- Print output to a file: set logging on
- Print data type of variable: ptype var
- Print symbol information: info symbol 0xff

**Print Virtual Memory**

Virtual memory may be printed with the `x` command:

```
(gdb) x/32xc 0x00007f3498000000
0x7f3498000000:   32 ' ' 0 '\000' 0 '\000' 28 '\034' 54 '6'
127 '\177' 0 '\000' 0 '\000'
0x7f3498000008:   0 '\000' 0 '\000' 0 '\000' -92 '\244' 52 '4'
127 '\177' 0 '\000' 0 '\000'
0x7f3498000010:   0 '\000' 0 '\000' 0 '\000' 4 '\004' 0 '\000'
0 '\000' 0 '\000' 0 '\000'
0x7f3498000018:   0 '\000' 0 '\000' 0 '\000' 4 '\004' 0 '\000'
0 '\000' 0 '\000' 0 '\000'
```

Another option is to dump memory to a file and then spawn an xxd process from within gdb to dump that file which is easier to read:

```
(gdb) define xxd
Type commands for definition of "xxd".
End with a line saying just "end".
>dump binary memory dump.bin $arg0 $arg0+$arg1
>shell xxd dump.bin
>shell rm -f dump.bin
>end
(gdb) xxd 0x00007f3498000000 32
0000000: 2000 001c 367f 0000 0000 00a4 347f 0000 ...6........4...
0000010: 0000 0004 0000 0000 0000 0004 0000 0000 .............
```

For large areas, these may be dumped to a file directly:

```
(gdb) dump binary memory dump.bin 0x00007f3498000000 0x00007f34a000000
```

Large VMAs often have a lot of zero'd memory. A simple trick to filter those out is to remove all zero lines:

```
$ xxd dump.bin | grep -v "0000 0000 0000 0000 0000 0000 0000 00000000" | less
```
Process Virtual Address Space

Gdb can query a core file and produce output about the virtual address space which is similar to /proc/$ {PID}/smaps, although it is normally a subset of all of the VMAs:

```
(gdb) info files
Local core dump file:
  `core.16721.dmp', file type elf64-x86-64.
  0x00007f3498000000 - 0x00007f34a0000000 is load51...
```

A GDB python script may be used to sum all of these address ranges:
https://raw.githubusercontent.com/kgibm/problemdetermination/master/scripts/gdb/gdbinfofiles.py

Debug a Running Process

You may attach gdb to a running process:

```
$ gdb ${PATH_TO_EXECUTABLE} ${PID}
```

This may be useful to set breakpoints. For example, to break on a SIGABRT signal:

```
(gdb) handle all nostop noprint noignore
(gdb) handle SIGABRT stop print noignore
(gdb) continue
```

# ... Reproduce the problem ...

```
Program received signal SIGABRT, Aborted.
[Switching to Thread 0x7f232df12700 (LWP 23949)]
0x00000000033a000d720 in sem_wait () from /lib64/libpthread.so.0
(gdb) ptype $_siginfo
type = struct {
  int si_signo;
  int si_errno;
  int si_code;
  union {
    int _pad[28];
    struct {...} _kill;...
  } _sifields;
}
(gdb) ptype $_siginfo._sifields._kill
```

```
type = struct {
    __pid_t si_pid;
    __uid_t si_uid;
}
```

```
(gdb) p $_siginfo._sifields._kill.si_pid
$1 = 22691
```

```
(gdb) continue
```

Next we can search for this PID 22691 and we'll find out who it is (in the following example, we see bash and the user name). If the PID is gone, then it is presumably some sort of script that already finished (you could create a background process that writes ps output to a file periodically to capture this):
$ ps -elf | grep 22691 | grep -v grep
 0 S kevin    22691  20866  0  80   0 - 27657 wait   08:16 pts/2    00:00:00
bash

Strictly speaking, you must first consult the signal number to know which union member to print above in `$_siginfo._sifields._kill`: [http://man7.org/linux/man-pages/man2/sigaction.2.html](http://man7.org/linux/man-pages/man2/sigaction.2.html)

**Shared Libraries**

Check if a shared library is stripped of symbols:

```
$ file $LIBRARY.so
```

Check the output for "stripped" or "non-stripped."

**glibc**

**malloc**

The default Linux native memory allocator on most distributions is Glibc malloc (which is based on ptmalloc and dlmalloc). Glibc malloc either allocates like a classic heap allocator (from sbrk or mmap’ed arenas) or directly using mmap, depending on a sliding threshold (M_MMAP_THRESHOLD). In the former case, the basic idea of a heap allocator is to request a large block of memory from the operating system and dole out chunks of it to the program. When the program frees these chunks, the memory is not returned to the operating system, but instead is saved for future allocations. This generally improves the performance by avoiding operating system overhead, including system call time. Techniques such as binning allows the allocator to quickly find a "right sized" chunk for a new memory request.

The major downside of all heap allocators is fragmentation (compaction is not possible because pointer addresses in the program could not be changed). While heap allocators can coalesce adjacent free chunks, program allocation patterns, malloc configuration, and malloc heap allocator design limitations mean that there are likely to be free chunks of memory that are unlikely to be used in the future. These free chunks are essentially "wasted" space, yet from the operating system point of view, they are still active virtual memory requests ("held" by glibc malloc instead of by the program directly). If no free chunk is available for a new allocation, then the heap must grow to satisfy it.

In the worst case, with certain allocation patterns and enough time, resident memory will grow unbounded. Unlike certain Java garbage collectors, glibc malloc does not have a feature of heap compaction. Glibc malloc does have a feature of trimming (M_TRIM_THRESHOLD); however, this only occurs with contiguous free space at the top of a heap, which is unlikely when a heap is fragmented.

Starting with glibc 2.10 (for example, RHEL 6), the default behavior was changed to be less memory efficient but more performant by creating per-thread arenas to reduce cross-thread malloc contention:

> Red Hat Enterprise Linux 6 features version 2.11 of glibc, providing many features and enhancements, including... An enhanced dynamic memory allocation (malloc) behaviour enabling higher scalability across many sockets and cores. This is achieved by assigning threads their own memory pools and by avoiding locking in some situations. The amount of
additional memory used for the memory pools (if any) can be controlled using the 
environment variables MALLOC_ARENA_TEST and MALLOC_ARENA_MAX. 
MALLOC_ARENA_TEST specifies that a test for the number of cores is performed once 
the number of memory pools reaches this value. MALLOC_ARENA_MAX sets the 
maximum number of memory pools used, regardless of the number of cores. 

After a certain number of arenas have already been created (2 on 32-bit and 8 on 64-bit, or the value 
explicitly set through the environment variable MALLOC_ARENA_TEST), the maximum number of 
arenas will be set to NUMBER

Glibc malloc does not make it easy to tell if fragmentation is the cause of process size growth, versus 
program demands or a leak. The malloc_stats function can be called in the running process to print free 
statistics to stderr. It wouldn't be too hard to write a JVMTI shared library which called this function 
through a static method or MBean (and this could even be loaded dynamically through Java Surgery). 
More commonly, you'll have a core dump (whether manually taken or from a crash), and the malloc 
structures don't track total free space in each arena, so the only way would be to write a gdb python 
script that walks the arenas and memory chunks and calculates free space (in the same way as 
malloc_stats). Both of these techniques, while not terribly difficult, are not currently available. In 
general, native heap fragmentation in Java program is much less likely than native memory program 
demands or a leak, so I always investigate those first (using techniques described elsewhere).

If you have determined that native heap fragmentation is causing unbounded process size growth, then 
you have a few options. First, you can change the application by reducing its native memory demands. 
Second, you can tune glibc malloc to immediately free certain sized allocations back to the operating 
system. As discussed above, if the requested size of a malloc is greater than 
M_MMAP_THRESHOLD, then the allocation skips the heaps and is directly allocated from the 
operating system using mmap. When the program frees this allocation, the chunk is un-mmap'ed and 
thus given back to the operating system. Beyond the additional cost of system calls and the operating 
system needing to allocate and free these chunks, mmap has additional costs because it must be zero-
filled by the operating system, and it must be sized to the boundary of the page size (e.g. 4KB). This 
can cause worse performance and more memory waste (ceteris paribus).

If you decide to change the mmap threshold, the first step is to determine the allocation pattern. This 
can be done through tools such as ltrace (on malloc) or SystemTap, or if you know what is causing 
most of the allocations (e.g. Java DirectByteBuffers), then you can trace just those allocations. Next, 
create a histogram of these sizes and choose a threshold just under the smallest yet most frequent 
allocation. For example, let's say you've found that most allocations are larger than 8KB. In this case, 
you can set the threshold to 8192:

    MALLOC_MMAP_THRESHOLD_=8192

Additionally, glibc malloc has a limit on the number of direct mmaps that it will make, which is 65536 
by default. With a smaller threshold and many allocations, this may need to be increased. You can set 
this to something like 5 million:

    MALLOC_MMAP_MAX_=5000000

These are set as environment variables in each Java process. Note that there is a trailing underscore on
these variable names.

You can verify these settings and the number and total size of mmaps using a core dump, gdb, and glibc symbols:

```
(gdb) p mp_
$1 = {trim_threshold = 131072, top_pad = 131072, mmap_threshold = 4096,
    arena_test = 0, arena_max = 1, n_mmaps = 1907812, n_mmaps_max =
    5000000,
    max_n_mmaps = 2093622, no_dyn_threshold = 1, pagesize = 4096,
    mmapped_mem = 15744507904, max_mmapped_mem = 17279684608, max_total_mem
    = 0,
    sbrk_base = 0x1e1a000 ""
```

In this example, the threshold was set to 4KB (mmap_threshold), there are about 1.9 million active mmaps (n_mmaps), the maximum number is 5 million (n_mmaps_max), and the total amount of memory currently mmap'ped is about 14GB (mmapped_mem).

There is also some evidence that the number of arenas can contribute to fragmentation.

**Ideas for dealing with fragmentation:**

1. Reduce the number and/or frequency of direct or indirect mallocs.
2. Create thread local caches of whatever is using the mallocs (e.g. DirectByteBuffers). Set the minimum size of the thread pool doing this equal to the maximum size to avoid thread local destruction.
3. Experiment with a limited MALLOC_ARENA_MAX (try 1 to see if there's any effect at first).
4. Experiment with MALLOC_MMAP_THRESHOLD and MALLOC_MMAP_MAX, carefully monitoring the performance difference.
5. Batch the frees together (e.g. with a stop-the-world type of mechanism) to increase the probability of free chunks coalescing.
6. Experiment with M_MXFAST
7. malloc_trim is rarely useful outside of academic scenarios. It only trims from the top of the main arena. First, most fragmentation is within an arena not at the top, and second, most programs heavily (and even predominantly) use the non-main arenas.

**How much is malloc'ed?**

Add mp_.mmapped_mem plus system_mem for each arena starting at main_arena and following the next pointer until next==&main_arena

```
(gdb) p mp_.mmapped_mem
$1 = 0
(gdb) p &main_arena
$2 = (struct malloc_state *) 0x3c95b8ee80
(gdb) p main_arena.system_mem
$3 = 413696
(gdb) p main_arena.next
$4 = (struct malloc_state *) 0x3c95b8ee80
```
Exploring Arenas

glibc provides malloc statistics at runtime through a few methods: mallinfo, malloc_info, and malloc_stats. mallinfo is old and not designed for 64-bit and malloc_info is the new version which returns an XML blob of information. malloc_stats doesn't return anything, but instead prints out total statistics to stderr (http://udrepper.livejournal.com/20948.html).

malloc trace

Malloc supports rudimentary allocation tracing:

Native Memory Leaks

eBPF

On kernel versions >= 4.1, eBPF may be used to dynamically attach to a process, set kernel hooks for native memory allocations, and periodically dump any stacks that do not have matching frees:

1. Install bcc and dependencies:
   https://github.com/iovisor/bcc/blob/master/INSTALL.md#packages
2. Clone bcc tools: git clone https://github.com/iovisor/bcc
3. Run the memleak script, specifying the process to watch, the interval in seconds, and, optionally, the number of iterations in seconds: bcc/tools/memleak.py -p ${PID} 30 10 > memleak_${PID}.txt
4. Analyze the stack output. For example:
   [19:41:11] Top 10 stacks with outstanding allocations:
   225144 bytes in 159 allocations from stack
   func1+0x16 [process]
   main+0x81 [process]
5. Background: https://raw.githubusercontent.com/iovisor/bcc/master/tools/memleak_example.txt

LinuxNativeTracker

The IBM Java team may be able to provide a custom build of Java through a PMR with the LinuxNativeTracker tool built-in.

Debug Symbols

In general, it is recommended to compile executables and libraries with debug symbols: "GCC, the GNU C/C++ compiler, supports ‘-g’ with or without ‘-O’, making it possible to debug optimized code. We recommend that you always use ‘-g’ whenever you compile a program." (https://www.sourceware.org/gdb/current/onlinedocs/gdb.html).

Alternatively, symbols may be output into separate files and made available for download to support engineers: http://www.sourceware.org/gdb/current/onlinedocs/gdb/Separate-Debug-Files.html

When compiling with GCC, consider -fno-omit-frame-pointer to ensure that frame pointers are not omitted so that backtraces are in tact.
Most Linux distributions have separate "debuginfo" packages that may be installed:

- RedHat Enterprise Linux: https://access.redhat.com/knowledge/solutions/9907

**Network**

On Linux, once a socket is listening, there are two queues: a SYN queue and an accept queue (controlled by the backlog passed to listen). Once the handshake is complete, a connection is put on the accept queue, if the current number of connections on the accept queue is less than the backlog. The backlog does not affect the SYN queue because if a SYN gets to the server when the accept queue is full, it is still possible that by the time the full handshake completes, the accept queue will have space. If the handshake completes and the accept queue is full, then the server's socket information is dropped but nothing sent to the client; when the client tries to send data, the server would send a RST. If syn cookies are enabled and the SYN queue reaches a high watermark, after the SYN/ACK is sent, the SYN is removed from the queue. When the ACK comes back, the SYN is rebuilt from the information in the ACK and then the handshake is completed.

**Hung Processes**

Gather and review (particularly the output of each kernel stack in /stack):

```bash
date
cat /proc/${PID}/status
date
cat /proc/${PID}/sched
date
cat /proc/${PID}/schedstat
date
cat /proc/${PID}/syscall
date
echo -en "/proc/${PID}/wchan="; cat /proc/${PID}/wchan
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/wchan; echo ""; done
date
echo -en "/proc/${PID}/stack="; cat /proc/${PID}/stack
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/stack; echo ""; done
date
echo -en "/proc/${PID}/syscall="; cat /proc/${PID}/syscall
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/syscall; echo ""; done
done
date
time
cat /proc/${PID}/wchan

echo -en "/proc/${PID}/stack="; cat /proc/${PID}/stack
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/stack; echo ""; done
done
date
time
cat /proc/${PID}/syscall
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/syscall; echo ""; done
done
date
time
cat /proc/${PID}/wchan

echo -en "/proc/${PID}/stack="; cat /proc/${PID}/stack
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/stack; echo ""; done
done
date
time
cat /proc/${PID}/syscall
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/syscall; echo ""; done
done
date
time
cat /proc/${PID}/wchan

echo -en "/proc/${PID}/stack="; cat /proc/${PID}/stack
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/stack; echo ""; done
done
date
time
cat /proc/${PID}/syscall
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/syscall; echo ""; done
done
date
time
cat /proc/${PID}/wchan

echo -en "/proc/${PID}/stack="; cat /proc/${PID}/stack
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/stack; echo ""; done
done
date
time
cat /proc/${PID}/syscall
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/syscall; echo ""; done
done
date
time
cat /proc/${PID}/wchan

echo -en "/proc/${PID}/stack="; cat /proc/${PID}/stack
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/stack; echo ""; done
done
date
time
cat /proc/${PID}/syscall
for i in /proc/${PID}/task/*; do echo -en "$i="; cat $i/syscall; echo ""; done
done
```

Review if number of switches is increasing:

```bash
PID=8939; PROC=sched; for i in /proc/${PID} /proc/${PID}/task/*; do echo -en "$i/PROC="; echo ""; cat $i/$PROC; echo ""; done | grep -e $PROC= -e nr_switches
```

**Kernel Dumps**

```
crash /var/crash/<timestamp>/vmcore /usr/lib/debug /lib/modules/<kernel>/vmlinux
```
Note that the <kernel> version should be the same that was captured by kdump. To find out which kernel you are currently running, use the `uname -r` command.

To display the kernel message buffer, type the `log` command at the interactive prompt.

To display the kernel stack trace, type the `bt` command at the interactive prompt. You can use `bt <pid>` to display the backtrace of a single process.

To display status of processes in the system, type the `ps` command at the interactive prompt. You can use `ps <pid>` to display the status of a single process.

To display basic virtual memory information, type the `vm` command at the interactive prompt. You can use `vm <pid>` to display information on a single process.

To display information about open files, type the `files` command at the interactive prompt. You can use `files <pid>` to display files opened by only one selected process.

kernel object file: A vmlinux kernel object file, often referred to as the namelist in this document, which must have been built with the `-g` C flag so that it will contain the debug data required for symbolic debugging.

When using the `fbt` provider, it helps to run through the syscall once with all to see what the call stack is and then hone in.


**Troubleshooting AIX**

**Request core dump (also known as a "system dump" for IBM Java)**

Additional methods of requesting system dumps for IBM Java are documented in the Troubleshooting IBM Java and Troubleshooting WAS chapters.

1. The gencore command pauses the process while the core is generated and then the process should continue. Replace `${PID}` in the following example with the process ID. You must have permissions to the process (i.e. either run as the owner of the process or as root) and the core `ulimit` must be unlimited (`$ulimit -c unlimited`) before starting the process. The size of the core file will be the size of the virtual size of the process (`ps VSZ`). If there is sufficient free space in physical RAM and the filecache, the core file will be written to RAM and then asynchronously written out to the filesystem which can dramatically improve the speed of generating a core and reduce the time the process is paused. In general, core dumps compress very well (often up to 75%) for transfer.

   `gencore ${PID} core.`date +%Y%m%d.%H%M%S`.dmp`
Signals
Signal mappings and detailed error codes may be found in /usr/include/sys/signal.h. A simpler listing may be performed with `kill -l`:

```shell
$ kill -l
1) HUP  14) ALRM  27) MSG  40) bad trap 53) bad trap
2) INT  15) TERM  28) WINCH  41) bad trap 54) bad trap
3) QUIT  16) URG  29) PWR  42) bad trap 55) bad trap
4) ILL  17) STOP  30) USR1  43) bad trap 56) bad trap
5) TRAP  18) TSTP  31) USR2  44) bad trap 57) bad trap
6) ABRT  19) CONT  32) PROF  45) bad trap 58) RECONFIG
7) EMT  20) CHLD  33) DANGER  46) bad trap 59) CPUFAIL
8) FPE  21) TTIN  34) VTALRM  47) bad trap 60) GRANT
9) KILL  22) TTOU  35) MIGRATE  48) bad trap 61) RETRACT
10) BUS  23) IO  36) PRE  49) bad trap 62) SOUND
11) SEGV  24) XCPU  37) VIRT  50) bad trap 63) SAK
12) SYS  25) XFSZ  38) ALRM1  51) bad trap
13) PIPE  26) bad trap  39) WAITING  52) bad trap
```

Find PID that owns a socket

Method 1
Install the optional `lsof` tool.

Method 2

```shell
$ netstat -Aan | grep "*.2[2,5].*LISTEN"
f1000e000531a3b8 tcp4 0 0 .22 .*. LISTEN
f1000e00040babb8 tcp4 0 0 .25 .*. LISTEN
```

```shell
$ rmsock f1000e000531a3b8 tcpcb
The socket 0xf1000e000531a008 is being held by proccess 9830644 (sshd).

$ rmsock f1000e00040babb8 tcpcb
The socket 0xf1000e00040ba808 is being held by proccess 6684754 (sendmail).
```

Method 3

1. Find the socket of interest with netstat -A and copy the first hexadecimal address which is the socket ID:
   ```bash
   # netstat -Aan | grep 32793
   f1000e00032203b8 tcp4 0 0 127.0.0.1.32793 .*. LISTEN
   ```

2. Send this socket ID into kdb:
   ```bash
   # echo "sockinfo f1000e00032203b8 tcpcb" | kdb | grep proc
   F1000F0A.00000000 F1000F0A10000000 pvproc+000000
   proc/fd: 65/8
   proc/fd: fd: 8
   pvproc+010400 65*kuxagent ACTIVE 0410058 0000001 0000008A84D5590 0 0030
   ```
3. Take the first hexadecimal address after ACTIVE and convert it to decimal:

```bash
# echo "hcal 0410058" | kdb | grep Value
Value hexa: 00410058  Value decimal: 4259928
```

4. Search for this PID in `ps`:

```bash
# ps -elf | grep 4259928
```

**Kernel Trace**

**Trace source of kill signal**

It may be useful to understand what PID is sending a kill signal to a process on AIX. You can use this kernel trace:

```
Login as root
# rm -rf /tmp/aixtrace; mkdir /tmp/aixtrace/; cd /tmp/aixtrace/
# trace -C all -a -T 10M -L 20M -n -j 134,139,465,14e,46c -o ./trc
... Reproduce the problem ... e.g. kill -3 7667754
# trcstop
# cp /etc/trcfmt .
# trcnm -a > trace.nm
# LDR_CNTRL=MAXDATA=0x80000000 gensyms > trace.syms
# LDR_CNTRL=MAXDATA=0x80000000 gennames -f > gennames.out
# pstat -i > trace.inode
# ls -al /dev > trace.maj_min2lv
```

Either zip and send these files to a PMR or analysis machine, or run these commands directly to process the trace:

```
# trcrpt -C all -r -o trc.tr trc
# trcrpt -C all -t trcfmt -n trace.nm -x -O
pid=on,tid=on,svc=on,exec=on,cpuid=on,PURR=on -o trc.txt trc.tr
```

Make sure the trace buffers did not wrap:

```
# grep WRAP trc.txt
```

If there are no results, then you're good; otherwise, if you see lines such as:

```
006   -1-   -1   -1   963.205627656
0.0029122602 963.205627656 TRACEBUFFER WRAPAROUND 0003
005   -4916246-   -1   4916246   113967573  963.205627656*
863.205627656 LOGFILE WRAPAROUND 0002
```

Then, either try increasing buffer sizes or reducing your test case or system load (or the tracepoints in -j).

Finally, search for the signal:

```
# grep -Ei "^14e|46c" trc.txt | grep -E "signal 3|SIGQUIT"
14E  ksh  0  10879036 62128373  28.157542500
0.128249  28.157542500 kill: signal SIGQUIT to process ?
java
```
The time of the signal is the ELAPSED_SEC column added to the date at the top of trc.txt:

```
# head -2 trc.txt
```

Wed Aug 21 05:10:28 2013

Thus the kill was sent at 05:10:56 by PID 10879036 (ksh). If this is a long running process, then you can reference ps.out for more details. The entry may not print the PID the signal was sent to (notice the question mark), but you should be able to figure that out based on other artifacts produced at that time such as javacores.

### Review Processor Affinity

To check for virtual processor affinity, we can use PerfPMR to gather kernel trace and then use curt. Run PerfPMR (example):

```
# perfpmr.sh 600
```

If you only want to collect processor affinity information, and you don't want everything else in PerfPMR, then you can collect just the kernel trace that's needed (this example is for 10 seconds):

```
# perfpmr.sh -x trace.sh 10
```

Process the PerfPMR output directory:

```
# perfpmr.sh -x trace.sh -r
```

Run curt on the trace:

```
# curt -i trace.tr -n trace.syms -est -r PURR -o curt.out
```

Open curt.out. The report is split up into system, per-CPU, and per-thread analysis. For each thread (section starts with "Report for Thread Id"), find the "processor affinity:" line.

```
# grep "processor affinity:" curt.out
```

The ideal affinity is 1.0 (meaning that the virtual processor is always going back to the same physical processor, thus maximizing cache hits, etc.) and the worst affinity is 0. Affinity may be low if a partition is above its entitlement and the shared processor pool does not have extra capacity or is in flux, because the partition will constantly have to take cycles from other processors.

Perform this before the performance problem occurs (under full load) and during the problem and compare the affinities. If affinity decreased during the problem, then the lack of entitlement may be making things worse. Be careful with cause and effect here: it's unlikely (though possible) that the decreased affinity in and of itself caused the problem, but instead was a secondary symptom that made things worse.

Processor affinity may be worse depending on the "spread" over the physical processors with a large number of configured virtual processors. Recent versions of AIX introduced processor folding which tries to optimize the use of the least number of virtual processors both to increase affinity and to decrease processor management overhead. Nevertheless, it may help to have the number of virtual processors not much higher than the entitled capacity or the effectively used capacity (see the processor folding link on how to calculate virtual processors).
Querying Queue Depth with Netstat

Example AIX netstat output:

```
$ netstat -an
Active Internet connections (including servers)
Proto Recv-Q Send-Q  Local Address          Foreign Address        (state)
tcp4       0      0  10.30.30.36.80       10.30.30.113.38378     ESTABLISHED
[...]
```

AIX has a somewhat strange format with the IP address followed by a period (instead of a colon) and the port number.

To find the queue depth of a server, first search for the local IP followed by the listening port of the server (it doesn’t matter which column it’s in because it’s impossible for a server to both listen on a port and use that same port as a client port for some other socket), and filter the output for sockets in ESTABLISHED or SYN_RECEIVED states. Sockets may also be in various closing states such as CLOSE_WAIT, TIME_WAIT, FIN_WAIT_1, and FIN_WAIT_2 (depending on which side closed its half of the socket first): these states can cause other types of bottlenecks but they are not related to queue depth.

For example, let’s say the web server above is listening on port 80:

```
$ netstat -an | grep -F 10.30.30.36.80 | grep -e ESTABLISHED -e SYN_RECEIVED | wc -l
14
```

Then, to get the queue depth, subtract this number from the maximum number of servers (e.g. MaxClients, WebContainer maximum size, etc.).

Debug Symbols

AIX: "Specifying -g will turn off all inlining unless you explicitly request it with an optimization option."

Use stabsplit to create separate symbol files:

Analyzing Native Memory with svmon

AIX, like all other modern operating systems, aggressively uses RAM as a file cache; therefore, it’s very common for the “free” RAM in AIX to show as very low. However, in general, this file cache can be pushed out of RAM to make space for program demands. Therefore, to understand “effective RAM usage”, you must subtract non-pinned filecache from used RAM.

aixmem.sh:

```
#!/bin/sh
PID=$1
INTERVAL=3600
echo "PID=$PID, INTERVAL=$INTERVAL"
```

Page 569
while ([ -d /proc/$PID ]); do
date
svmon -G
svmon -r -m -P $PID
kill -3 $PID
sleep $INTERVAL
done

Run the following to make the script executable:
chmod a+x aixmem.sh

Start the script (and replace $PID with the target Java process ID):
nohup ./aixmem.sh $PID > nativemem.txt 2>&1 &

This runs at an interval of 30 minutes and has a very low overhead (assuming you have not changed the default behavior of `kill -3` to only produce a javacore).

After enough data has been captured, kill the script:
kill $(ps -ef | grep nativemem.sh | grep -v grep | awk '{print $2}')</n

The following command line snippet on Linux (where I ran the command; I haven’t tested on AIX, but it should be easy to use with ksh) may be used to analyze the output of the AIX memory script above and display the effectively used RAM (i.e. RAM usage - file cache):

grep -e TZ -e "'^memory" -e "'^in use" nativemem.txt | \n  while read line; do \n    read line2;
    read line3;
    echo $line;
    rambytesused="$(echo "$\{line2\}" | awk '{print $3}' | echo "$(cat -)*4096" | bc)");
    rambytesfilecache="$(echo "$\{line3\}" | awk '{print $5}' | echo "$(cat -)*4096" | bc)");
    printf "Total RAM usage (bytes): %s\n" "$\{rambytesused\}";
    printf "File cache RAM usage (bytes): %s\n" "$\{rambytesfilecache\}";
    printf "Effectively used RAM (bytes): %s\n\n" "$\{rambytesused\}-\$\{rambytesfilecache\}" | bc);\n  done

Native Memory Leaks

Restart the process with the MALLOCDEBUG envvar to track un-freed mallocs. This may have a significant performance overhead:
export MALLOCDEBUG=report_allocations,stack_depth:3

Reproduce the problem and stop the process gracefully. A report is produced in stderr with each un-freed allocation:

Allocation #0: 0x3002FD60
Allocation size: 0x2A0
Allocated from heap: 0
Allocation traceback:
0xD01DC934 malloc
0xD01288AC init_malloc
0xD012A1F4 malloc
0xD04DFF34 __pth_init

Snapshots of this data may also be captured with a core dump (see above) and the dbx $(malloc) command (see below).

With Java, careful of using stack depths greater than 3:

"The stack depth of 3 provides only a limited stack trace. However, the use of larger stack depths with a Java application can cause crashes because the debug malloc facility does not understand the stack frames used for JIT compiled code."

Run format_mallocdebug_op.sh to aggregate and summarize the stacks:

Example output:

```
ZIP_Put_In_Cache
readCEN
calloc_common
malloc

################################
533676 bytes leaked in 127 Blocks
################################
```

**dbx**

Analyze a core file:

```
$ dbx ${PATH_TO_EXECUTABLE} ${PATH_TO_CORE}
```

To load shared libraries from a particular folder, user -p:

```
$ dbx -p ./ ${PATH_TO_EXECUTABLE} ${PATH_TO_CORE}
```

If you see the following warning:

```
warning: The core file is not a fullcore. Some info may not be available.
```

Then the core is probably truncated. As recommended in the Java documentation, enable fullcore and reproduce the issue

```
# chdev -l sys0 -a fullcore='true' -a pre430core='false'
```

**Tips**

Type `help $COMMAND` to print the summary and options of a command. For example:

```
(dbx) help proc
```
proc [raw] [cred | cru | rlimit | ru | sigflags | signal]

Display process information. "raw" displays in raw hex format. "cred" shows credentials, "cru", "ru" and "rlimit" resource info, "sigflags" and "signal" information about signals and handlers.

Command output may be redirected to files in the same directory. For example:

```
(dbx) coremap > coremap.txt
```

proc

```
`proc` prints general process information. For example:

```
(dbx) proc
pi_pid:       9306144   pi_sid:       10354784
pi_ppid:      9961578   pi_pgrp:      204
pi_uid:       204      pi_suid:      204
pi_thcount:   342      pi_cpu:       0
pi_start:     Tue Dec 9 06:09:20 2014
pi_tsize:     0x0000000000013eeb   pi_dsize:   0x000000003ba99c00...
```
```

thread

```
`thread` prints a list of all native threads. The thread preceded with "->" is the current thread:

```
(dbx) thread
thread  state-k     wchan            state-u    k-tid mode held scope function
      $t1     run       0xf1000f0a1015c140 blocked  96535077    k   no   sys _event_sleep...
      $t190   run                           running   123339089   k   no   sys pollset_poll
      >$t286   run                           running   96272413    k   no   sys genSystemCoreUsingGencore...
```
```

Native Stacks

On AIX, to calculate the total native memory used by native stacks:

```
(dbx) thread info
```

For example, under "stack storage," the native stack size is next to size=:

```
thread  state-k     wchan            state-u    k-tid mode held scope function
      $t1     run                  blocked  28999789    u   no   sys _event_sleep...
      stack storage:              base = 0x2df23000   size = 0x1fffc830
```
```

where

`where` prints a native stack of the current thread. For example:
To print the stack of a particular thread, reference the number with $t#:

```
(dbx) where $t190
Thread $t190
  warning: Thread is in kernel mode, not all registers can be accessed.
pollset_poll(??, ??, ??, ??) at 0x90000000014e56c
pollStatus() at 0x900000005e8ca30
```

`map` prints a list of all loaded modules. For example:

```
(dbx) map
Entry 1:
  Object name: ./opt/BPM/8.5/WebSphere/AppServer/java/jre/bin/java
  Text origin: 0x100000000000
  Text length: 0x17236
  Data origin: 0x1001000012b
  Data length: 0x194d
  File descriptor: 0x5
...
Entry 64:
  Object name: ./usr/lib/libc.a
  Member name: shr_64.o
  Text origin: 0x900000000000c80
  Text length: 0x43b3bf
  Data origin: 0x9001000a00007e0
  Data length: 0x11d8a0
  File descriptor: 0x83
```

`coremap` prints a list of all memory mappings. For example:

```
Mapping: Shared Memory (size=0x280000000)
  from (address): 0xa0000000000000 - 0xa000000280000000
  to (offset) : 0x421680be - 0x2c21680be
  in file     : core.20141209.175356.9306144.0002.dmp
```

In the above example, the virtual address range is from 0xa00000000000000 to 0xa000002800000000 (which is of length 0x280000000 reported in the first line), and the raw data may be found in the file `core.20141209.175356.9306144.0002.dmp` in the range 0x421680be to 0x2c21680be. We can verify
this by dumping the first 2 words of the virtual address:

\[
\text{(dbx) 0xa00000000000000/2X}
\]
\[
0x0a00000000000000: 00000100 13737f3c
\]

This matches dumping the same bytes from the core file at the offset:

\[
\text{\$ od -N 8 -x core.20141209.175356.9306144.0002.dmp +0x421680be}
\]
\[
421680be 0000 0100 1373 7f3c
\]

**Print memory**

An address followed by a slash, a number, and a format character may be used to print raw memory. For example:

\[
\text{(dbx) 0x10000000000/8X}
\]
\[
0x0000010000000000: 01f70005 51c013e3 00000000 0003a516
\]
\[
0x0000010000000010: 00781182 000015af 010b0001 00000000
\]

The help command for this is `help display`.

For null-terminated strings, simply type the address followed by /s.

For character bytes, use /c. For example:

\[
\text{(dbx) 0x20000000000/8c}
\]
\[
0x0000020000000000: 'A' 'B' 'C' 'D' 'E' 'F' '1' '2'
\]

**malloc**

`malloc` prints a summary of the malloc subsystem. For example:

\[
\text{(dbx) malloc}
\]

The following options are enabled:

```
Implementation Algorithm........ Default Allocator (Yorktown)
```

```
Statistical Report on the Malloc Subsystem:
Heap 0
    heap lock held by................. pthread ID 0x1001000ee90
    bytes acquired from sbrk()....... 1000964480
    bytes in the freespace tree...... 125697184
    bytes held by the user............ 875267296
    allocations currently active..... 56012
    allocations since process start.. 65224401
```

```
The Process Heap
    Initial process brk value........ 0x0000010010001a80
    current process brk value........ 0x000001004ba99c00
    sbrk()s called by malloc......... 7180
```

**corefile**

`corefile` prints information about a loaded core file. For example:
(dbx) corefile
Process Name:  /opt/IBM/WebSphere/AppServer/java/jre/bin/java
Version:       500
Flags:         FULL_CORE | CORE_VERSION_1 | MSTS_VALID | UBLOCK_VALID |
               USTACK_VALID | LE_VALID
Signal:        
Process Mode:  64 bit

Related Commands

• Print all printable strings and their hex offsets (at least N printable characters followed by a null, default N is 4):
  $ strings -t x $CORE_FILE
• Print all bytes in both hexadecimal and character from the core file starting at offset 0x988 and only show 100 bytes:
  $ od -N 100 -xc $CORE_FILE +0x988
• Alternatively:
  $ od -v -A x -N 100 -j 0x2B521000 -t xc $CORE_FILE

Troubleshooting z/OS

z/OS often refers to a date in the form: 09.210. In this case, 09 are the last two digits of the year. 210 means it is the 210th day of year 2009; in this example, July 29, 2009.

MODIFY Commands

z/OS supports modify commands which request diagnostic data from an address space:

• Request javacores on servants:
  MODIFY $CONTROLLER,JAVACORE
• Request stacks on servants:
  MODIFY $CONTROLLER,STACKTRACE

Console Dump

Take a console dump from the operator console with the title $X of the address space with ID $Y, responding to the operator console identifier $Z returned by the DUMP command (replace X, Y, and Z):

  DUMP COMM=($X)
  R $Z,ASID=$Y,CONT
  R $Z SDATA=(PSA,CSA,LPA,LSQA,RGN,SQA,SUM,SWA,TRT,ALLNUC,GRSQ),END

Troubleshooting IBM i

Gathering Javacores using WRKJVMJOB

Gathering Javacores is covered in the IBM i Operating System chapter.
Troubleshooting Windows

Common Commands

- Query Windows Version: ver
- Query host name: hostname
- Show all processes: tasklist /svc
- Kill process: taskkill /PID %PID%

Request thread dump

See Troubleshooting IBM Java - in particular, you may use Java Surgery to take thread dumps on IBM Java.

Request core dump (also known as a "system dump" for IBM Java)

Additional methods of requesting system dumps for IBM Java are documented in the Troubleshooting IBM Java and Troubleshooting WAS chapters.

1. On Windows, start Task Manager, right click on the process, click Create Dump File. You can find the right process by adding the PID column, and finding the PID from SystemOut.log or the %SERVER%.pid file in the logs directory.
2. procdump: https://docs.microsoft.com/en-us/sysinternals/downloads/procdump

Minidump versus Full User Mode Dump

The name "minidump" is misleading, because the largest minidump files actually contain more information than the "full" user-mode dump. (https://msdn.microsoft.com/en-us/library/windows/hardware/ff552212(v=vs.85).aspx)

Determine the Current Working Directory of a Process

2. Unzip and start procexp.exe
3. Find the java process, right click and click Properties:
4. Find the value of "Current directory":

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU</th>
<th>Private Bytes</th>
<th>Wo</th>
</tr>
</thead>
<tbody>
<tr>
<td>armovc.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>audiograd.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BESClient.exe</td>
<td>2.20</td>
<td>12,180 K</td>
<td></td>
</tr>
<tr>
<td>BESClientHelper.exe</td>
<td>5.968</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BESClientUI.exe</td>
<td>0.03</td>
<td>10,594 K</td>
<td></td>
</tr>
<tr>
<td>blnsrv.exe</td>
<td>0.04</td>
<td>2,252 K</td>
<td></td>
</tr>
<tr>
<td>c4ebrcq.exe</td>
<td></td>
<td>11,736 K</td>
<td></td>
</tr>
<tr>
<td>ccSvcHist.exe</td>
<td>0.38</td>
<td>36,528 K</td>
<td></td>
</tr>
<tr>
<td>ccSvcHist.exe</td>
<td>&lt; 0.01</td>
<td>5,348 K</td>
<td></td>
</tr>
<tr>
<td>cmd.exe</td>
<td></td>
<td>2,652 K</td>
<td></td>
</tr>
<tr>
<td>conhost.exe</td>
<td></td>
<td>3,884 K</td>
<td></td>
</tr>
<tr>
<td>csrss.exe</td>
<td>0.15</td>
<td>2,156 K</td>
<td></td>
</tr>
<tr>
<td>csrss.exe</td>
<td>0.01</td>
<td>2,008 K</td>
<td></td>
</tr>
<tr>
<td>dwm.exe</td>
<td></td>
<td>2,084 K</td>
<td></td>
</tr>
<tr>
<td>explorer.exe</td>
<td>0.03</td>
<td>25,840 K</td>
<td></td>
</tr>
<tr>
<td>Interrupts</td>
<td>0.37</td>
<td>0 K</td>
<td></td>
</tr>
<tr>
<td>lsasr.exe</td>
<td></td>
<td>8,024 K</td>
<td></td>
</tr>
<tr>
<td>msrsrv.exe</td>
<td></td>
<td>12,320 K</td>
<td></td>
</tr>
<tr>
<td>process.exe</td>
<td>50.37</td>
<td>184,496 K</td>
<td></td>
</tr>
<tr>
<td>process.exe</td>
<td></td>
<td>184,496 K</td>
<td></td>
</tr>
<tr>
<td>services.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>smss.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>spoolsv.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sppsvc.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sqlwriter.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>svchost.exe</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Click on "java.exe" and select "Properties..." from the context menu.
Determine the File Locations of stdout and stderr


2. Unzip and open a command prompt to the directory with the unzipped files and run the tool to list all open handles for all processes:
   ```
   > cd "C:\Downloads\Handle"
   > handle.exe
   ```

3. Search the output for the java.exe section for the target process ID. For example:
   ```
   java.exe pid: 4852 [...]
   C: File (R--) C:\work\stderr.txt
   [...]
   ```

4. The left-most value is a hexadecimal number representing the file handle. Stdin is A, Stdout is B, and Stderr is C (https://msdn.microsoft.com/en-us/library/windows/desktop/ms683231(v=vs.85).aspx); however, we have observed that sometimes stdout is not B, but instead there are multiple handles in the D to FF range (unclear why).

Error: No buffer space available (maximum connections reached?)

This error can occur particularly around socket operations. The error is translated from the Winsock error code WSAENOBUFFS, 10055 (https://msdn.microsoft.com/en-us/library/windows/desktop/ms740668(v=vs.85).aspx). The most common cause of this error is that Windows is configured for the default maximum of 5,000 in-use ports. This can be monitored by watching netstat or Perfmon and can be changed with the MaxUserPort registry parameter (https://technet.microsoft.com/en-us/library/cc739819(WS.10).aspx).

A more advanced cause for this error is non-paged pool exhaustion. The paged and nonpaged pools are areas of memory for certain Windows kernel-mode allocations such as the Windows kernel itself (e.g.
sockets, socket buffers, etc.), device drivers, etc.
nonpaged pool is particularly important as "it is the availability of nonpaged pool that determines how
many processes, threads, and other such objects can be created." (https://support.microsoft.com/kb/108449)
If these pools are exhausted, this can lead to crashes, poor
performance, application problems, and paging.

If the system is using the /3GB mode, this comes at a cost of taking memory away from the kernel,
including paged and non-paged pools.

To determine if this is the proximate cause, use Perfmon to monitor the Memory\Pool Nonpaged Bytes
counter. If this is hitting the server's nonpaged pool limit (within a few MB since Perfmon is sampling
on an interval), then this is the cause of the problem. However, this proximate cause may not be the
root cause since the nonpaged pool exhaustion may be due to a nonpaged pool leak. A nonpaged pool
leak can be determined using Microsoft's poolmon.exe (https://support.microsoft.com/kb/177415).

To increase kernel memory, lower the user-mode address space limit (/USERVA=X)
(https://support.microsoft.com/kb/316739). The /3GB switch is effectively the same as /USERVA=3072
(https://support.microsoft.com/kb/810371); for example, /USERVA=2800. This parameter would be
used instead of /3GB. The documentation is not clear on how the additional space is allocated to the
nonpaged pool and to what limits -- monitor your current and maximum nonpaged pool sizes with
process explorer and work with Microsoft support to properly tune this value.

**windbg**

Useful commands:

- Command help: .hh %COMMAND%
- Interrupt a long running command: Ctrl+Break
- Clear output: .cls
- Write output to file: .logopen %SOMEFILE%
- Show modules the process has in memory: lmf
- Confirm if symbols are loaded for MODULE.dll: ld MODULE
  - If you see "Default to export symbols for MODULE.dll," then symbols were not
    found or did not match.
- Write output to file: .logopen %SOMEFILE%
- List all virtual memory regions: laddress
- Virtual memory info: !address -summary
- List all native heaps: !heap -s
- List details of a heap (ID from first column in !heap -s): !heap -stat -h <Heap ID>
- Given a UserPtr and EXE has gflags +ust, dump stack: !heap -p -a <UserPtr>
- Was gflags set: !gflag
- Dump memory at an arbitrary address: db 0x123...
- Show where symbols are found: lm
- Show checksum: !lmi MODULE
- Module information: !dh MODULE
- Show why symbols can't be found: !sym noisy; .reload /f
- Try to load symbols for a particular module (take the name from lmf): ld %MODULE%
- See if DLL and PDB match: !chksym MODULE
• NOTE: If a DLL is compiled in “Release” mode (which most are), then you will not see line numbers or parameters even if the PDB has private symbols.

Symbols

The symbol path is a semicolon delimited list of directories containing symbol (PDB) files. A special case path is of the form srv*%DIR%*%WEBSERVER% which specifies an HTTP(S) symbol server. This is most often used to download Windows symbols from Microsoft. If the %DIR% does not exist, it is created.

Simple symbol path with just kernel symbols:

```
0:000> .sympath srv*C:\symbols*http://msdl.microsoft.com/download/symbols
0:000> .reload /f
```

For example, a common path which includes WebSphere and Java symbols:

```
0:000> .sympath C:\Program Files\IBM\WebSphere\AppServer\lib\native\win\x86_64;C:\Program Files\IBM\WebSphere\AppServer\java\8.0\jre\bin\compressedrefs;C:\Program Files\IBM\WebSphere\AppServer\java\8.0\jre\bin\j9vm;C:\Program Files\IBM\WebSphere\AppServer\bin;srv*C:\symbols*http://msdl.microsoft.com/download/symbols
0:000> .reload /f
```

It is common to see a checksum warning when loading modules:

```
*** WARNING: Unable to verify checksum for ...
```

In general, this warning can be safely disregarded. If you would like to resolve the warning, run `editbin /release module.dll`: [https://msdn.microsoft.com/en-us/library/tst6zb25.aspx](https://msdn.microsoft.com/en-us/library/tst6zb25.aspx)

To display detail symbol loading information:

```
0:000> !sym noisy
noisy mode - symbol prompts on
0:000> .reload /f
```

Check if symbols were correctly loaded for a module by searching for MATCH:

```
0:000> !chksym module
module.dll...
pdb: ...
module.pdb

module.pdb
pdb sig: EDD67653-11E7-483C-8D6D-E629DC820CC1
age: 2

MATCH: module.pdb and module.dll
If symbols were not loaded, you may various errors such as "sig MISMATCH." In the following example, the PDB file has a signature of E98..., whereas the DLL has a signature of 0.

0:000> !chksym mymodule

mymodule.dll
  Timestamp: 54415058
  SizeOfImage: 7000
    pdb sig: 0
    age: 0

Loaded pdb is ...\mymodule.pdb

getClasses.pdb
  pdb sig: E98AF532-F9C8-4205-9E83-0512360C6C93
  age: 0

sig MISMATCH: mymodule.pdb and mymodule.dll

Symbol loading options may be displayed:

0:000> .symopt
Symbol options are 0x30237:

Symbol loading options may be updated. For example, to use SYMOPT_LOAD_ANYTHING:

0:000> .symopt +0x40
Symbol options are 0x30277:

After changing symbol loading options, you may need to reload symbols with `.reload /f` or reload a particular module with `.reload /f module.dll`

**Process and Thread Info**

Show process information:

0:000> |

List threads:

0:000> ~

Current thread:

0:000> ~.

Crashing thread:

0:000> ~#

Display when a thread was created and how much user and kernel time it has used:

0:000> .ttime
Created: Fri Oct  6 05:13:34.194 2017 (UTC - 4:00)
Kernel:  0 days 0:00:00.046
User:    0 days 0:00:00.062
Current thread stack:
0:000> kn

Stacks for all threads:
0:000> ~*kn

To find the crashing thread from an IBM Java core dump, use jdmpview or IDDE:

> !gpinfo | grep Failing
Failing Thread: !j9vmthread 0x409e000
Failing Thread ID: 0xa048 (41032)

Take the part in bold above and change the current thread in windbg and then print the stack:
0:000> ~~[a048]s
0:000> kn

To switch to a particular frame and show its registers:
0:000> .frame /r 11

To display locals:
0:000> dv

To dump details about a variable:
0:000> dt somevariable

To disassemble a function:
0:000> uf MODULE!SYMBOL

Note that the registers on the frame that crashed might be strange because they may have been hijacked by the signal handler to create the core dump, so you'll have to check what the registers were on that frame in the javacore:::

1XHREGISTERS   Registers:
2XHREGISTER     RDI: 0000D07C00000000
2XHREGISTER     RSI: 0000000000079D0
2XHREGISTER     RAX: 0000000000000000[...]

If needed, replace "2XHREGISTER" with "r" and ":" with "=" for all register lines except XMM* and apply them in windbg on the .frame:
0:000> r RDI= 0000D07C00000000
0:000> r RSI= 0000000000079D0
0:000> r RAX= 0000000000000000 [...] 

Crash Dump Analysis
0:000> !analyze -v
Virtual Address Space (!address)

Use !address to print all virtual memory allocations. Only Windows symbols are required to execute this:

```
windbg.exe > File > Open Crash Dump... > Select .dmp file > Save Information for Workspace? = No
0:000> .sympath srv*C:\symbols\*http://msdl.microsoft.com/download/symbols
0:000> .reload /f
0:000> .logopen c:\windbg.txt
0:000> !address

<table>
<thead>
<tr>
<th>BaseAddress</th>
<th>EndAddress+1</th>
<th>RegionSize</th>
<th>Type</th>
<th>State</th>
<th>Protect</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>0`00000000</td>
<td>0`00010000</td>
<td>0`00010000</td>
<td>MEM_FREE</td>
<td>PAGE_NOACCESS</td>
<td>Free</td>
<td></td>
</tr>
<tr>
<td>* 0`00010000</td>
<td>0`00020000</td>
<td>0`00010000</td>
<td>MEM_MAPPED</td>
<td>PAGE_READWRITE</td>
<td>Heap [Handle: 10000]...</td>
<td></td>
</tr>
</tbody>
</table>
```

The following script may be used to analyze the output of !address:

https://raw.githubusercontent.com/kgibm/problemdetermination/master/scripts/windows/windbgaddress.s.pl

Native memory heaps (!heap)

Use `!heap -s` to print statistics on all native memory heaps. Only Windows symbols are required to execute this:

```
windbg.exe > File > Open Crash Dump... > Select .dmp file > Save Information for Workspace? = No
0:000> .sympath srv*C:\symbols\*http://msdl.microsoft.com/download/symbols
0:000> .reload /f
0:000> !heap -s

LFH Key                   : 0x000000911bae555a
Termination on corruption : ENABLED
Heap Flags Reserv Commit Virt Free List UCR Virt
Lock  Fast   (k) (k) (k) (k) length  blocks
cont. heap
--------------------------------------------------------------------------------------------------------------------------
0000000000260000 00000002 3593912 1920496 3593912 855294 6524 1953 66
72ea  LFH
External fragmentation 44 % (6524 free blocks)
Virtual address fragmentation 46 % (1953 uncommitted ranges)
000000000010000 00008000 64 4 64 2 1 1 0
0...
```

In general, "External fragmentation" is the most interesting fragmentation number and calculates how much free space is available between active allocations. In this example, 0.44 * 3,593,912 = 1.5GB.

Printing all heap segments for a particular heap identifier will show the address ranges of virtual allocations:
0:000> !heap -m -h 260000
Index Address Name Debugging options enabled
1: 00260000
Segment at 0000000000260000 to 000000000035f000 (000ff000 bytes committed)
Segment at 00000000001e70000 to 00000000001f6f000 (000ff000 bytes committed)...

Printing detailed heap statistics will show a histogram of free block sizes:
0:000> !heap -f -stat -h 260000
0: Heap 0000000000260000
Flags 00000002 - HEAP_GROWABLE
Reserved memory in segments 5342216 (k)
Commited memory in segments 1609368 (k)
Virtual bytes (correction for large UCR) 1653528 (k)
Free space 576170 (k) (5196 blocks)

<table>
<thead>
<tr>
<th>Range (bytes)</th>
<th>Default heap</th>
<th>Front heap</th>
<th>Unused bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Busy</td>
<td>Free</td>
<td>Busy</td>
</tr>
<tr>
<td>0 - 1024</td>
<td>2253</td>
<td>2599</td>
<td>3610</td>
</tr>
<tr>
<td>1024 - 2048</td>
<td>2192</td>
<td>8</td>
<td>74</td>
</tr>
<tr>
<td>2048 - 3072</td>
<td>132</td>
<td>40</td>
<td>48</td>
</tr>
</tbody>
</table>

The output of !address will also print the heap for each of the virtual allocations. If investigating exhaustion of some space (e.g. underneath 4GB), then review the heaps used in that space.

Dump virtual memory

The `db` command accepts a start and end address:

0:000> db 0xffb1d000 0xffb24000
00000000`ffb1d000  00 00 00 00 00 00 00 00-36 dd f3 85 a6 da fc
00  ........6.......
00000000`ffb1d010  10 70 ac 57 00 00 00 00-30 20 65 1d 00 00 00 00 .p.W....0
e.....

Native Stack Sizes

It appears that Windows will allocate 1MB of virtual stack space for every thread even if a program requests less: "The default stack reservation size used by the linker is 1 MB." (https://msdn.microsoft.com/en-ca/library/windows/desktop/ms686774(v=vs.85).aspx).

For example, on recent versions of IBM Java, the default maximum stack size (-Xss) is 256KB or 512KB (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.win.80.doc/diag/appendices/cmdline/Xss.html); however, a userdump in windbg showed:

0:000> !address
BaseAddress EndAddress+1 RegionSize Usage...
* 0`f0560000 0`f05a0000 0`00040000 Stack [338.55e0;~1450]
* 0`f0660000 0`f06a0000 0`00040000 Stack [338.5be8;~1450]
For example, the stack @ 0xf0560000 has a region size of 256KB; however, the next stack doesn't start until 756KB later.

Thread stacks may also be printed with the !threads command:

```
0:000> !threads
```

<table>
<thead>
<tr>
<th>Index</th>
<th>TID</th>
<th>TEB</th>
<th>StackBase</th>
<th>StackLimit</th>
<th>DeAlloc</th>
<th>StackSize</th>
<th>ThreadProc...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1637</td>
<td>0000000000004c24</td>
<td>0x000007fff0736000</td>
<td>0x00000000f15c0000</td>
<td>0x00000000f15bc000</td>
<td>0x00000000f14c0000</td>
<td>0x0000000000004000</td>
<td>0x0</td>
</tr>
</tbody>
</table>

Total VM consumed by thread stacks 0x1997f000

In this example, we can also see that StackBase-DeAlloc = 1MB.

Module

List all loaded modules:

```
0:000> lmf
```

<table>
<thead>
<tr>
<th>start</th>
<th>end</th>
<th>module name</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000`00400000</td>
<td>00000000`0042f000</td>
<td>java C:...\java\bin\java.exe</td>
</tr>
<tr>
<td>00000000`77750000</td>
<td>00000000`7786f000</td>
<td>kernel32 C:\Windows\System32\kernel32.dll...</td>
</tr>
</tbody>
</table>

Dump flags

The information in a dump is controlled with the MINIDUMP_TYPE enumeration:

This may be queried on an existing dump with .dumpdebug (see the "Flags" line):

```
0:000> .dumpdebug
```

```
----- User Mini Dump Analysis
MINIDUMP_HEADER:
    Version         A793 (6804)
    NumberOfStreams 7
    Flags           2
    0002 MiniDumpWithFullMemory
```

Frame Pointer Omission (FPO)

There is a compiler optimization called Frame Pointer Omission (/Oy) which speeds up function calls:

However, this breaks stack walker. Microsoft Visual C++ 2005 enabled this optimization by default but it was later disabled by default in Visual C++ 2005 SP1, therefore, it is a best practice to avoid FPO.

Check if FPO is used by an EXE or DLL:

```
dumpbin.exe /fpo %MODULE%
```

If there are “FPO Data” lines, then it is used.
Debug and Release Modules

In general, Microsoft compilers provide two compilation configurations: Debug and Release. These supply different compiler and linker parameters such as optimization levels and linking to different libraries. These are simply default sets of configurations, and it is possible to change all of the flags of a "release" build to make it a "debug" build. The salient point here is that a module such as a DLL or EXE may perform worse with Debug optimization flags (whether due to using the Debug configuration or by explicitly using /Od, /MTd, etc.). It is not easy to check a final DLL or EXE to understand its optimization level. One technique is to see if it links with debug versions of commonly used libraries such as VC, MFC, or ATL. For example, use a dependency walker to see if the module depends on MSVCRRTD.DLL. However, this is not proof as it is possible to create a fully optimized module that links to a debug version of one of these libraries. Another technique is to search the module for references to functions only called with _DEBUG or NDEBUG #defined, such as assert.

Symbols

Symbols match hexadecimal addresses to human readable descriptions from the original source code, such as 0x12345678 is the function foo. Symbols are required when analyzing native artifacts such as process core dumps (userdumps). Windows EXEs and DLLs do not contain symbol information, but instead the symbols are placed into PDB files, normally with the same name and in the same directory as the EXE or DLL. PDBs should always be built, even for release-optimized modules:

"Generating PDB files for release executables does not affect any optimizations, or significantly alter the size of the generated files... For this reason, you should always produce PDB files, even if you don't want to ship them with the executable."


To generate symbols, add the /Zi compiler flag: https://msdn.microsoft.com/en-us/library/958x11bc.aspx

While not generally recommended, if you would like to name the PDB file something other than MODULE.pdb, use /Fd: https://msdn.microsoft.com/en-us/library/9wst99a9.aspx

If you have separate compile and link steps, in addition to the compiler /Zi flag, you must also add the linker /DEBUG flag: https://msdn.microsoft.com/en-us/library/xe4t6fc1.aspx. Note that the term "DEBUG" in this context has nothing to do with /D _DEBUG, /Od, /MTd or other "Debug" compiler configurations, but instead simply "puts the debugging information into a program database (PDB)." In fact, the linker will update the PDB file created by the compiler through the /Zi flag, so both are required.

Often, symbols will not be distributed with EXEs and DLLs simply to reduce the size of installer packages. Windows itself does not ship with PDBs. However, if the additional size of PDBs of EXEs and DLLs is marginal, then we recommend that you ship the PDBs with the EXEs and DLLs. IBM Java ships PDBs with each build (http://www-01.ibm.com/support/docview.wss?uid=swg1IV50063), and WAS is working on adding PDBs to all of its DLLs (http://www-01.ibm.com/support/docview.wss?uid=swg1PM85208).

While Windows symbols can be downloaded for a particular build (see retail symbols in https://msdn.microsoft.com/en-us/windows/hardware/gg463028), in general, it is better to use the Microsoft Symbol Server which will download any matching symbols on demand. If you are
debugging a core dump from a machine other than your own that is running a different version of Windows, then using the Microsoft Symbol Server is the best approach:

The common Microsoft debugging tools use the SymSrv technology if you provide the correct symsrv syntax in the _NT_SYMBOL_PATH environment variable. These tools automatically include whatever you provide in the variable as the symbol path.

You can set this variable as a system variable or as a user environment variable. To do this from the desktop, right-click My Computer, and then click Properties. On the Advanced tab, click Environment Variables.

You can also set this variable temporarily at a command prompt. In this way, all applications that you start through the command prompt inherit this setting.


Example:

```bash
> set _NT_SYMBOL_PATH = SRV*c:\symbols*http://msdl.microsoft.com/download/symbols
> windbg
```

In the above example, symbols downloaded from the symbol server will be cached locally in the c:\symbols\ folder for future use.

For WAS, assuming WAS in C:\Program Files\IBM\WebSphere\AppServer, an example sympath would be:

```
C:\Program Files\IBM\WebSphere\AppServer\bin;C:\Program Files\IBM\WebSphere\AppServer\java\jre\bin;C:\Program Files\IBM\WebSphere\AppServer\java\jre\bin\j9vm;C:\Program Files\IBM\WebSphere\AppServer\bin;srv*C:\symbols*http://msdl.microsoft.com/download/symbols
```

If the machine does not have internet access, you can run "symchk /om" to get a list of symbols that are needed, then download that set of symbols from a machine that does have internet access using "symchk /im" and then copy the symbols over.

If you do not want to ship PDB symbols, then you should still save PDBs for each build and make them available to support engineers. Ideally, these can be offered through a custom symbol server:

"Setting up a symbol server on your own local network is as simple as creating a file share on a server... To add, delete or edit files on a symbol server share, use the symstore.exe tool." (https://msdn.microsoft.com/en-us/library/ee416588(v=vs.85).aspx)

For external use, an HTTP symbol server can be setup using IIS on top of the symbol directory.

You can find all the PDBs in a directory using *.pdb in explorer.exe search or from a command line (https://technet.microsoft.com/pt-pt/library/cc754900(v=ws.10).aspx):

```bash
for /r %i in (*) do @echo %ftzai | findstr pdb
```

> "C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x64\symchk.exe" /v module.dll /s DIRECTORYWITHPDB
Success: DBGHELP: private symbol & lines
Failure: FAILED – built without debugging information

See if a PDB file is corrupt:

> "C:\Program Files (x86)\Windows Kits\8.0\Debuggers\x66\pdbcopy.exe" somefile.pdb test.pdb -p
Can't open pdb file...Error: EC_CORRUPT

Desktop Heap

Windows Desktop Heaps are limited areas of virtual memory allocated for programs that use functions in user32.dll: "The desktop heap stores certain user interface objects, such as windows, menus, and hooks. When an application requires a user interface object, functions within user32.dll are called to allocate those objects. If an application does not depend on user32.dll, it does not consume desktop heap." (https://blogs.msdn.microsoft.com/b/ntdebugging/archive/2007/01/04/desktop-heap-overview.aspx, https://support.microsoft.com/kb/184802).

Troubleshooting Solaris

Mapping LWP ID to Java thread

It's often useful to map an LWP ID (for example, reported in prstat -L) to a Java thread. The pstack command may be used to print all native stack traces along with the LWP ID:

```
# prstat -mvLp 5598 5 2
  PID USERNAME USR SYS TRP TFL DFL LCK SLP LAT VCX ICX SCL SIG PROCESS/LWPID
  5598 root  78 2.8 0.1 0.0 1.7 1.3 7.0 8.7 135 502 3K 2 java/2
  5598 root 12 0.0 0.0 0.1 0.0 85 0.0 2.7 54 59 124 0 java/10...

# pstack 5598
5598: /opt/IBM/WAS855/AppServer/java_1.7_32/bin/java Play
-----------------
lwp# 2
  thread# 2  --------------------
fbc895a0 * java/util/StringTokenizer.nextToken()Ljava/lang/String;
[compiled] +74 (line 691)
  fbc895a0 * java/crypto/Cipher.a(Ljava/lang/String;)[Ljava/lang/String;+55
  fbcad2d10 * java/crypto/Cipher.b(Ljava/lang/String;)[Ljava/lang/String;)
[compiled] +2
  fbc99494 *
*java/crypto/Cipher.getInstance(Ljava/lang/String;)Ljava/crypto/Cipher;
[compiled] +2
  fbcabc29c * *Play.main([Ljava/lang/String;)V [compiled] +61 (line 39)
  fbc0021c * StubRoutines (1)
  fe5b635c
  __1cJJavaCallsCall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCallArguments_pnGThread_v_ (fe0f0d8, fe0f7e0, e, 27800, f5cb0, d79a4fc8) + 3a0
  fe65be7c jni_CallStaticVoidMethod (27928, d79a4fc8, 21240, e, 27800,
```
ff117e5c) + 678
    ff361bd8 JavaMain (fe66537c, 28e6c, 27928, ff387370, ff0f261c, fe65b804) + 740
    ff2c5238 _lwp_start (0, 0, 0, 0, 0, 0)...

Request core dump
1. The gcore command pauses the process while the core is generated and then the process should continue. Replace ${PID} in the following example with the process ID. You must have permissions to the process (i.e. either run as the owner of the process or as root). The size of the core file will be the size of the virtual size of the process (ps VSZ). If there is sufficient free space in physical RAM and the filecache, the core file will be written to RAM and then asynchronously written out to the filesystem which can dramatically improve the speed of generating a core and reduce the time the process is paused. In general, core dumps compress very well (often up to 75%) for transfer. ([http://docs.oracle.com/cd/E36784_01/html/E36870/gcore-1.html](http://docs.oracle.com/cd/E36784_01/html/E36870/gcore-1.html))

$ gcore ${PID}

2. If gcore does not start to write the core dump immediately, it may be hung waiting to acquire control of the process. If this does not succeed for some time, try instead with the -F option.

3. If none of the other options work, you may crash the process which should process a core dump using one of:
   $ kill -6 ${PID}
   $ kill -11 ${PID}

Debug Symbols
"To compile optimized code for use with dbx, compile the source code with both the -O (uppercase letter O) and the -g options... The -g0 (zero) option turns on debugging and does not affect inlining of functions." ([http://docs.oracle.com/cd/E19205-01/819-5257/gevhr/index.html](http://docs.oracle.com/cd/E19205-01/819-5257/gevhr/index.html))


Troubleshooting HP-UX

32-bit Native OutOfMemoryErrors
You may increase the user virtual address space from 2GB to 3GB (at the cost of less space to the kernel for things like network buffers) with:

    chatr +q3p enable ${PATH_TO_JAVA}

You can check if this is enable with:

    chatr ${PATH_TO_JAVA}
    ...
    third quadrant private data space enabled
**gdb/wdb**

When the process is hung, attach to the PID, for example:

```
/opt/langtools/bin/gdb /opt/IBM/WebSphere/AppServer/java/bin/IA64W/java 24072
```

Then run `thread apply all bt`

### Print full command line of running program

HP-UX does not provide a tool (such as "ps") to print the full command line of a running program (no equivalent of Solaris /usr/ucb/ps). The -x parameter of ps only prints the first 1024 characters, which is often insufficient for Java programs:

> Only a subset of the command line is saved by the kernel; as much of the command line will be displayed as is available... The value of DEFAULT_CMD_LINE_WIDTH should be between 64 and 1020.

You can attach to a process using gdb/wdb and print argc/argv. First, we attach to a process by passing in the location of java (which you can get from ps -elfx) followed by the PID (note that the process will be completely paused until you detach gdb):

```
$ /opt/langtools/bin/gdb /opt/IBM/WebSphere/AppServer/java/bin/IA64W/java 24072
```

__argc and __argv are global variables that we can access, so let's first see how many arguments there are:

```
(gdb) print __argc
$1 = 3
```

In this example, we have 3 arguments. Next, we know that argv is a pointer to a list of pointers, each with one of the program arguments, so we print that many addresses at the location of argv (i.e. replace 3 with your value of argc):

```
(gdb) x/3a __argv
0x9ffffffffffff950:     0x9ffffffffffff9e8      0x9ffffffffffffa19
0x9ffffffffffff960:     0x9ffffffffffffa24
```

Each of these addresses is a pointer to a null-terminated string, so we print each using the s option:

```
(gdb) x/s 0x9ffffffffffff9e8
0x9ffffffffffff9e8:      "/opt/IBM/WebSphere/AppServer/java/bin/IA64W/java"
(gdb) x/s 0x9fffffffffffffa19
0x9fffffffffffffa19:      "HelloWorld"
(gdb) x/s 0x9fffffffffffffa24
0x9fffffffffffffa24:      "testarg"
```

Don't forget to "detach" to continue the process.

Although the Java jps command with the -v parameter is no better, at least you can use jps -m to map PID to WAS server name.

If you are using an Itanium system, the following caliper command prints the full command line. This
will have some overhead as it is gathering a flat profile of sampled process instructions for 1 second, but it is presumably more lightweight (and more user-friendly) than gdb:

```
/opt/caliper/bin/caliper fprof --process=root --attach $PID --duration 1 | grep Invocation:
```

And here's a one-line command that runs the above on all java PIDs:

```
for i in `ps -elfx | grep java | grep -v grep | awk '{print $4}'`; do echo $i; /opt/caliper/bin/caliper fprof --process=root --attach $i --duration 1 | grep Invocation: ;done;
```

### Troubleshooting Java

- [Troubleshooting IBM Java](#)
- [Troubleshooting Oracle Java](#)

### Request Heap Dump

There are many ways to request a heap dump depending on your Java vendor (and further depending on your operating system and WAS profile, detailed within each Java vendor's section):

- [Request System Dump on IBM Java](#)
- [Request HPROF Heap Dump on Oracle Java](#)

### Excessive Direct Byte Buffers

Excessive native memory usage by java.nio.DirectByteBuffers is a classic problem with any generational garbage collector such as gencon (which is the default starting in IBM Java 6.26/WAS 8), particularly on 64-bit. DirectByteBuffers (DBBs) ([http://docs.oracle.com/javase/6/docs/api/java/nio/ByteBuffer.html](http://docs.oracle.com/javase/6/docs/api/java/nio/ByteBuffer.html)) are Java objects that allocate and free native memory. DBBs use a PhantomReference which is essentially a more flexible finalizer and they allow the native memory of the DBB to be freed once there are no longer any live Java references. Finalizers and their ilk are generally not recommended because their cleanup time by the garbage collector is non-deterministic.

This type of problem is particularly bad with generational collectors because the whole purpose of a generational collector is to minimize the collection of the tenured space (ideally never needing to collect it). If a DBB is tenured, because the size of the Java object is very small, it puts little pressure on the tenured heap. Even if the DBB is ready to be garbage collected, the PhantomReference can only become ready during a tenured collection. Here is a description of this problem (which also talks about native classloader objects, but the principle is the same):

If an application relies heavily on short-lived class loaders, and nursery collections can keep up with any other allocated objects, then tenure collections might not happen very frequently. This means that the number of classes and class loaders will continue increasing, which can increase the pressure on native memory... A similar issue can arise with reference objects (for example, subclasses of java.lang.ref.Reference) and objects with finalize() methods. If one of these objects survives long enough to be moved into tenure space before becoming unreachable, it could be a long time before a tenure collection runs...
and "realizes" that the object is dead. This can become a problem if these objects are holding on to large or scarce native resources. We've dubbed this an "iceberg" object: it takes up a small amount of Java heap, but below the surface lurks a large native resource invisible to the garbage collector. As with real icebergs, the best tactic is to steer clear of the problem wherever possible. Even with one of the other GC policies, there is no guarantee that a finalizable object will be detected as unreachable and have its finalizer run in a timely fashion. If scarce resources are being managed, manually releasing them wherever possible is always the best strategy. (http://www.ibm.com/developerworks/websphere/techjournal/1106_bailey/1106_bailey.htm)

Essentially the problem boils down to either:

1. There are too many DBBs being allocated (or they are too large), and/or
2. The DBBs are not being cleared up quickly enough.

It is very important to verify that the volume and rate of DBB allocations are expected or optimal. If you would like to determine who is allocating DBBs (problem #1), of what size, and when, you can run a DirectByteBuffer trace. Test the overhead of this trace in a test environment before running in production.

If you would like to clear up DBBs more often (problem #2), there are a few options:

1. Use -XX:MaxDirectMemorySize=$bytes
   Specifying MaxDirectMemorySize will force the DBB code to run System.gc() when the sum of outstanding DBB native memory would be more than $bytes. This option may have performance implications. When using this option with IBM Java, ensure that -Xdisableexplicitgc is not used. The optimal value of $bytes should be determined through testing. The larger the value, the more infrequent the System.gcs will be but the longer each tenured collection will be. For example, start with -XX:MaxDirectMemorySize=1024m and gather throughput, response time, and verbosegc garbage collection overhead numbers and compare to a baseline. Double and halve this value and determine which direction is better and then do a binary search for the optimal value.

2. Explicitly call System.gc. This is generally not recommended. When DBB native memory is freed, the resident process size may not be reduced immediately because small allocations may go onto a malloc free list rather than back to the operating system. So while you may not see an immediate drop in RSS, the free blocks of memory would be available for future allocations so it could help to "stall" the problem. For example, Java Surgery can inject a call to System.gc into a running process: https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=7d3dc078-131f-404c-8bd4-68b3b9dd07a

3. One common cause of excessive DBB allocations with WebSphere Application Server is the default WAS WebContainer channelwritetype value of async. See the WAS HTTP section for more details.

In most cases, something like -XX:MaxDirectMemorySize=1024m (and ensuring -Xdisableexplicitgc is not set) is a reasonable solution to the problem.

A system dump or HPROF dump may be loaded in the IBM Memory Analyzer Tool & the IBM Extensions for Memory Analyzer DirectByteBuffer plugin may be run to show how much of the DBB
native memory is available for garbage collection. For example:

  => Sum DirectByteBuffer capacity available for GC: 1875748912 (1.74 GB)
  => Sum DirectByteBuffer capacity not available for GC: 72416640 (69.06 MB)

Java Surgery

There is an experimental technique called Java surgery which uses the JavaLateAttach API
(http://docs.oracle.com/javase/6/docs/technotes/guides/attach/index.html) to inject a JAR into a running
process and then execute various diagnostics:
https://www.ibm.com/developerworks/community/groups/service/html/communityview?
communityUuid=7d3dc078-131f-404c-8b4d-68b3b9ddd07a

This was designed initially for Windows because it does not usually have a simple way of requesting a
thread dump like `kill -3` on Linux. Java Surgery has an option with IBM Java to run the
com.ibm.jvm.Dump.JavaDump() API to request a thread dump (Oracle Java does not have an
equivalent API, although Java Surgery does generally work on Oracle Java):

  $ java -jar surgery.jar -pid 16715 -command JavaDump

Native Java Agents

JVMTI is the latest and is available in Java 5 and replaces JVMDI and JVMP. JVMDI and JVMPI are
fully deprecated in Java 6. When JVMPI is enabled, the JVM looks for the exported method
JVM_OnLoad in the binary. When JVMTI is enabled, the JVM looks for the exported method
Agent_OnLoad in the binary.

JVMP is specified using the -Xrun... command line option

JVMTI is specified using the -agentlib... or -agentpath... command line options

Java Serialization

Java serialization stream protocol:
http://docs.oracle.com/javase/7/docs/platform/serialization/spec/protocol.html

The eye catcher for a serialized Java object in a byte stream is 0xAC 0xED (STREAM_MAGIC).

Verbose Garbage Collection

Verbose garbage collection may be enabled with the Memory MBean:
Troubleshooting IBM Java

IBM Java Versions

IBM Java has different releases with various synonymous names:

- IBM Java Version 5, a.k.a. 1.5
- IBM Java Version 6, a.k.a. 1.6
- IBM Java Version 6 Release 1, a.k.a. 6.0.1, a.k.a. 1.6 build 2.6, a.k.a. J9 VM 2.6, a.k.a. 6R1, a.k.a. 626
- IBM Java Version 7, a.k.a. 1.7
- IBM Java Version 7 Release 1, a.k.a. 7.0.1, a.k.a. 1.7 build 2.7
- IBM Java Version 8

In addition, each release has service releases which are fixpack upgrades, normally with APARs, but sometimes with feature enhancements, for example SR1. Finally, service releases themselves may have fix packs, for example FP1.

An example IBM Java version commonly seen is: IBM Java Version 7R1 SR1 FP1

**Thread Dump (javacore.txt)**

IBM Java can produce a javacore.txt file, also called a javadump:
dump_interpret.html. An example file name is javacore.20140930.025436.9920.0003.txt.
Staring with IBM Java 5, you may see threads in a javacore which are in Conditional Wait (CW) state that you would expect to be Runnable (R). This is by design: http://www-01.ibm.com/support/docview.wss?uid=swg21413580; however, starting with Java 7.1 SR2, Java 7.0 SR8, and Java 6.1 SR8 FP2, such threads are reported as Runnable and the internal state is reported in the vmstate field (http://www-01.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.lnx.70.doc/diag/tools/javadump_tags_threads.html).

In general, javacores are very low overhead. They usually take no more than a few hundred milliseconds to produce. However, there are known defects in IBM Java that cause the entire JVM to freeze when requesting a javacore. These are usually caused by race conditions and are more likely the more javacores that are taken. Therefore, there is some risk in taking javacores in production, and this risk is proportional to the number of javacores taken. Before taking a lot of javacores, ensure that you have fixes installed for the most common of these hangs: http://www-01.ibm.com/support/docview.wss?uid=swg1IZ84925 and http://www-01.ibm.com/support/docview.wss?uid=swg1IV66662.

Request Thread Dump

Additional methods of requesting thread dumps are documented in the Troubleshooting WAS chapter.

1. On POSIX operating systems, by default, the command `kill -3 ${PID}` will request a thread dump.
3. On recent versions of IBM Java, use Java Surgery:
   $ java -jar surgery.jar -pid ${PID} -command JavaDump
4. The trace engine may be used to request a thread dump on method entry and/or exit:
   https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/trace_options_trigger.html. The following example JVM argument requests a thread dump when the Example.trigger() method is called:
   
   -Xtrace:maximal=mt,trigger=method{com/ibm/example/Example.trigger,javadump}

By default, a user requested javacore has the request=exclusive option which asks for exclusive access before executing the javacore. However, unlike other agents, if exclusive access cannot be granted, the javacore creation will still proceed with some data excluded for safety. There is a known APAR IV68447/PI32843 where heap corruption may occur if a javacore runs during a garbage collection cycle.

There are some older known issues of requesting javacores causing the JVM to hang: http://www-01.ibm.com/support/docview.wss?uid=swg1IZ89711

Process Limits (ulimits)

On recent versions of IBM Java and applicable operating systems, a javacore includes a section with
the current ulimit values of the process:

<table>
<thead>
<tr>
<th>Limit Type</th>
<th>Type</th>
<th>Soft Limit</th>
<th>Hard Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Limits (in bytes except for NOFILE and NPROC)</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>RLIMIT_AS</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_CORE</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_CPU</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_DATA</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_FSIZE</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_LOCKS</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_MEMLOCK</td>
<td>65536</td>
<td>65536</td>
<td>65536</td>
</tr>
<tr>
<td>RLIMIT_NOFILE</td>
<td>8192</td>
<td>8192</td>
<td>8192</td>
</tr>
<tr>
<td>RLIMIT_NPROC</td>
<td>213234</td>
<td>213234</td>
<td>213234</td>
</tr>
<tr>
<td>RLIMIT_RSS</td>
<td>unlimited</td>
<td>unlimited</td>
<td>unlimited</td>
</tr>
<tr>
<td>RLIMIT_STACK</td>
<td>8388608</td>
<td>8388608</td>
<td>8388608</td>
</tr>
<tr>
<td>RLIMIT_MSGQUEUE</td>
<td>819200</td>
<td>819200</td>
<td>819200</td>
</tr>
<tr>
<td>RLIMIT_NICE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RLIMIT_RTPRIO</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>RLIMIT_SIGPENDING</td>
<td>213234</td>
<td>213234</td>
<td>213234</td>
</tr>
</tbody>
</table>

When requesting a system dump using the IBM system dump mechanism, the JVM will ensure that the RLIMIT_CORE hard limit is used: [http://www-01.ibm.com/support/docview.wss?uid=swg21222437](http://www-01.ibm.com/support/docview.wss?uid=swg21222437)

**NATIVEMEMINFO**

In recent versions of IBM Java, the NATIVEMEMINFO section summarizes native memory allocations that the JVM has made: [http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/java_dump_tags_nativememinfo.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/java_dump_tags_nativememinfo.html)

<table>
<thead>
<tr>
<th>Section</th>
<th>NATIVEMEMINFO subcomponent dump routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMUSER</td>
<td>JRE: 4,786,464,960 bytes / 14237 allocations</td>
</tr>
<tr>
<td>MEMUSER</td>
<td>+-VM: 4,734,576,408 bytes / 11959 allocations</td>
</tr>
<tr>
<td>MEMUSER</td>
<td>+-Classes: 130,832,328 bytes / 5225 allocations</td>
</tr>
<tr>
<td>MEMUSER</td>
<td>+-Memory Manager (GC): 4,388,855,680 bytes / 1502 allocations</td>
</tr>
<tr>
<td>MEMUSER</td>
<td>+-Java Heap: 4,294,967,296 bytes / 1 allocation...</td>
</tr>
</tbody>
</table>
On 64-bit Java, the "Unused <32bit allocation regions: 6,708,704 bytes" line summarizes how much native memory is free in the pooled region allocations underneath 4GB.

If -Dcom.ibm.dbgmalloc=true is specified, then additional information will be added to NATIVEMEMINFO including native memory used by the Zip/Jar SDK code:

```
4MEMUSER | | +-Zip: 5,913,128 bytes / 2271 allocations
4MEMUSER | | +-Wrappers: 64,320 bytes / 193 allocations
5MEMUSER | | | +--Malloc: 64,320 bytes / 193 allocations
```

**Virtual Memory Layout**

A virtual memory layout of native memory allocations made by IBM Java may be created by simply listing the start and end addresses of all IBM Java native memory segments (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/javadump_tags_meminfo.html):

```
$ grep 1STSEGMENT javacore*.txt | awk '{print $3,$5}' | sort
```

This list may be useful to correlate to an operating system virtual memory layout to figure out what is allocated by IBM Java versus native allocations outside of IBM Java.

**Accumulated CPU Time**

Compare accumulated CPU time between threads across two javacores (replace the first two lines with the javacore file names):

```
JAVACORE1=javacore.20171117.145059.26621.0001.txt; 
JAVACORE2=javacore.20171117.145108.26621.0003.txt; 
join -a 1 -a 2 
  <(
    grep -e '3XMTHREADINFO ' -e 3XMCPUTIME "${JAVACORE1}" | 
    grep -v 'Anonymous native thread' | 
    sed '$!N;s/\n/ /' | 
    sed 's/3XMTHREADINFO.*J9VMThread://g' | 
    sed 's/.*CPU usage total://g' | 
    sed 's/ secs.*//g' | 
    sort 
  )
  <(
    grep -e '3XMTHREADINFO ' -e 3XMCPUTIME "${JAVACORE2}" | 
    grep -v 'Anonymous native thread' | 
    sed '$!N;s/\n/ /' | 
    sed 's/3XMTHREADINFO.*J9VMThread://g' | 
    sed 's/.*CPU usage total://g' | 
    sed 's/ secs.*//g' | 
    sort 
  ) | 
  awk '{ printf "%s %.9f\n", $0, $3-$2 }' | sort -nr -k 4
```

**Thread States**

The Javacore.txt thread dump shows the state of each thread at the time of the dump; most commonly,
R for runnable, CW for conditional wait, B for blocked, and P for parked. It has been a common confusion since IBM Java version 5 that threads which are effectively running (R) are actually reported as waiting (CW). This is because the JVM uses a cooperative mechanism to try to quiesce running threads for the duration of the Javacore to reduce the chances of problems creating the javacore itself ([http://www-01.ibm.com/support/docview.wss?uid=swg21413580](http://www-01.ibm.com/support/docview.wss?uid=swg21413580)). Tools such as IBM TMDA naively report the thread dump state without taking this into account:

<table>
<thead>
<tr>
<th>Name</th>
<th>State</th>
<th>Thread Name</th>
<th>State</th>
<th>Java Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Waiting on condition</td>
<td>main</td>
<td>Waiting on condition</td>
<td>at Loop.main(Loop.java:)</td>
</tr>
<tr>
<td>JIT Compiler</td>
<td>Waiting on condition</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Signal</td>
<td>Runnable</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Gc Slave</td>
<td>Waiting on condition</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Gc Slave</td>
<td>Waiting on condition</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Gc Slave</td>
<td>Waiting on condition</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

HTTP://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.diagnostics.60/diag/tools/java
dump_tags_threads.html

However, starting with Java 8, Java 7, Java 6.1, and Java 6 SR16 FP4, the javacore.txt file now reports these thread states as runnable, and moves the "true" state into the `vmstate` field: "Threads that were running Java code when the javacore was triggered have a Java thread state of R (Runnable) and an internal VM thread state of CW (Condition Wait)." ([http://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.diagnostics.60/diag/tools/java
dump_tags_threads.html](http://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.diagnostics.60/diag/tools/java
dump_tags_threads.html))

Heapdumps and system dumps

A heapdump contains information on the Java heap. This is used for investigating OutOfMemoryErrors, tuning Java heap usage, etc. On IBM Java, historically, a heapdump was equivalent to an IBM Portable Heapdump (PHD) file. A PHD heapdump is written by code in IBM Java and is generally limited to object reference analysis. Recently, IBM Java has pushed a new strategic direction to use system dumps instead of PHD heapdumps. A system dump is equivalent to the operating system process memory dump (Unix=core, Windows=dump, z/OS=DUMP, etc.). System dumps are written by the operating system. In essence, system dumps are a superset of PHD heapdumps. Not only do they include the Java heap, but they also include object memory (for example, the actual value of a String, etc.), which brings them to parity with HotSpot HPROF heapdumps. Additionally, system dumps include more detailed thread information (including some of the Java stack frame locals on each stack frame, which can be incredibly useful, such as finding out which database SQL query is executing), more accurate garbage collection root information, native memory
Starting in IBM Java 626 (WAS 8.0.0.2), a system dump has been added for the first OutOfMemoryError. Thus, the default has changed to produce a PHD heapdump, javacore, snap file, and a system dump on OOM (http://www-01.ibm.com/support/docview.wss?uid=swg21584396).

In older versions of IBM Java, the jextract tool was required to post-process a system dump. This was cumbersome and time consuming. Starting with Java 5 >= SR12 (WAS >= 6.1.0.33), Java 6 >= SR9 (WAS >= 7.0.0.15), Java 626 (WAS 8), DTFJ-based tools such as the IBM Memory Analyzer Tool (MAT) and IBM Interactive Diagnostic Data Explorer (IDDE) can read a system dump directly, just like a PHD heapdump (ftp://aix.software.ibm.com/software/isa/isa410/production/metadata/com.ibm.dtfj.feature_1.6.0.201106211324/featureDesc.html). Jextract may still be useful for investigating native memory information (because jextract will also gather native libraries from the filesystem), but in general, a system dump is now as easy to use as a PHD heapdump.

Unfortunately, most customers on Unix operating systems are still configured with constrained ulimits which truncate system dumps, making them usually useless. It is critical that you properly configure Unix systems for full core dumps:


System dumps usually compress to 25% of original size using zip, gzip, etc.

For the best system dump performance, ensure significant free physical memory so that the operating system can write it to RAM and then asynchronously flush to disk.

To analyze both heapdumps and system dumps, see the IBM Memory Analyzer Tool chapter.

**Portable Heap Dump (PHD)**

In general, IBM Java uses two formats for heapdumps: IBM Portable Heapdump (PHD) and an operating system dump. The latter is a superset of the former.

The operating system dump is simply a core dump of the virtual address space (Unix=core, Windows=userdump, z/OS=SYSDUMP) of the process. In older versions of IBM Java, the JVM's jextract tool was required to be run on an operating system dump before it could be analyzed. Starting with Java 5 >= SR12, Java 6 >= SR9, and later Java releases, jextract is not necessary because IBM has created file readers for operating system dumps for all operating systems on which IBM Java runs (http://www.ibm.com/developerworks/websphere/techjournal/1109_supauth/1109_supauth.html). Tools such as the IBM Memory Analyzer Tool use the IBM Diagnostic Tool Framework for Java API to read the heapdump from jextracted ZIPs or operating system dumps.

An IBM PHD file contains basic information about the Java heap such as the graph of relationships between objects and their size. An operating system dump is a superset of a PHD heap dump and includes everything about the process; thus, in general, it will be larger and take longer to produce than an IBM PHD file. An operating system dump is usually compressible down to 25% of its original size.
for transportation.

**Request heap dump**

Additional methods of requesting heap dumps are documented in the Troubleshooting WAS chapter.

1. Use `-Xdump:heap:user` to take one on kill `-3/Ctrl+Break`
2. Extract a PHD heapdump from a system dump using the heapdump command in jdpmpview or IDDE
4. From within the IBM Memory Analyzer Tool: File > Acquire Heap Dump
5. On recent versions of IBM Java, use Java Surgery:
   
   ```
   $ java -jar surgery.jar -pid ${PID} -command HeapDump
   ```
6. The trace engine may be used to request a heap dump on method entry and/or exit:
   The following example JVM argument produces a heap dump when the Example.trigger() method is called:
   
   ```
   -Xtrace:maximal=mt,trigger=method{com/ibm/example/Example.trigger,heapdump}
   ```

**System Dumps (core.dmp)**

**Request system dump**

Additional methods of requesting system dumps are documented in the Troubleshooting Operating Systems and Troubleshooting WAS chapter.

1. Automatically produced on a crash
2. Starting with Java 6.26 SR1, a system dump is produced on the first OutOfMemoryError:
3. In earlier versions of Java, a system dump may be produced on OOM with:
   
   -Xdump:heap:none
   -Xdump:java+system:events=systhrow,filter=java/lang/OutOfMemoryError,range=1..4,request=exclusive+serial+prepwalk
5. IBM Health Center can acquire a dump: Monitored JVM > Request a dump > System Dump
6. The IBM Java Dump Agent "system" can take a system dump on various events ([http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/diag/tools/dumpagents_system.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.zos.80.doc/diag/tools/dumpagents_system.html)). See Table 2 in Debugging from Dumps ([http://www.ibm.com/developerworks/library/j-memoryanalyzer/#N1026B](http://www.ibm.com/developerworks/library/j-memoryanalyzer/#N1026B)). For example, the following will create a core dump when the Example.bad method throws a NullPointerException:

```
-Xdump:system:events=throw,range=1..1,request=serial+exclusive+prepwalk,filter=java/lang/NullPointerException#com/ibm/example/Example.bad
```

7. The trace engine may be used to request a system dump on method entry and/or exit: [http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/trace_options_trigger.html](http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/trace_options_trigger.html). The following example JVM argument produces a system dump when the Example.trigger() method is called:

```
-Xtrace:maximal=mt,trigger=method{com/ibm/example/Example.trigger,sysdump}
```


```
$ java -jar surgery.jar -pid ${PID} -command SystemDump
```

9. From within the IBM Memory Analyzer Tool: File > Acquire Heap Dump

### Exclusive-access for System Dumps

One of the main issues with requesting system dumps is that the default dump agent for system dumps does not request exclusive access: [https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/dump_agents_defaults.html](https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/dump_agents_defaults.html)

This is required because some GPFs, aborts, and other conditions need to take system dumps without requesting exclusive access (for example, if there is a bug within the GC itself that causes a crash).

The problem is that if a system dump is requested while a garbage collection is running, this normally means that the system dump will be unusable by tools such as Memory Analyzer Tool. The garbage collector is modifying core data structures and moving pointers and references which will utterly confuse memory analysis tools. The API com.ibm.jvm.Dump.SystemDump which is used by most mechanisms to request system dumps (e.g. wsadmin, Liberty server dump, etc.) uses the default dump agent which means it does not request exclusive access.

On newer versions of IBM Java, the exclusive option may be passed to the triggerDump API call: [https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/dump_agents_request.html](https://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.vm.80.doc/docs/dump_agents_request.html)

On older versions of IBM Java, there aren't many good options to ensure that requesting a system dump requests exclusive access to avoid such a situation. The most obvious option would be to create a system dump on the user event (kill -3) and request exclusive access there; but, in general, the user event should be used for lightweight diagnostics such as thread dumps and it should not be used for
heapdumps or system dumps.

The best option is to create a dump agent which requests a system dump with the exclusive option when a diagnostic exception is thrown and then use a tool such as Java surgery to inject a small JAR into the JVM that throws such an exception. First, set the following JVM option:

```
-Xdump:system:events=throw,filter=com/ibm/rdci/surgery/builtin/commands/CustomException1,request=exclusive+serial
```

Then use [Java Surgery](http://www.ibm.com) to attach and throw the exception:

```
java -jar surgery.jar -command ThrowException -pid ${PID}
```

**jextract**

Recent versions of IBM Java do not require running jextract on the core dump for memory analysis (Java 5 >= SR12, Java 6 >= SR9, etc.). The reason is that the dump readers (DTFJ) are able to read operating system core dump files directly using DirectDumpReader (DDR) technology.

However, jextract is useful for investigating native memory issues because jextract will gather the java executable and native libraries which may be loaded into a debugger along with the core dump.

Jextract -interactive runs on the core file itself, not the jextracted ZIP. Example output:

```
# jextract -interactive core.20100624.110917.7576.0001.dmp
Loading dump file...
Read memory image from core.20100624.110917.7576.0001.dmp
Jextract interactive mode.
Type '!j9help' for help.
Type 'quit' to quit.
(Commands must be prefixed with '!')
>
!findallcallsites (or !dumpallsegments)
Searching for all memory block callsites...
Finished search. Bytes scanned: 4294958661
<p>| total alloc | largest |</p>
<table>
<thead>
<tr>
<th>blocks</th>
<th>bytes</th>
<th>bytes</th>
<th>callsite</th>
</tr>
</thead>
<tbody>
<tr>
<td>+--------+-------+-------</td>
<td>----------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>11 common/j9nls.c:427</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>176</td>
<td>176 ParallelGlobalGC.cpp:162...</td>
<td></td>
</tr>
</tbody>
</table>
```

**System dumps on Linux**

On Linux, when IBM Java requests the system dump, it forks itself and then kills the forked child process:

```
Linux does not provide an operating system API for generating a system dump from a running process. The JVM produces system dumps on Linux by using the fork() API to start an identical process to the parent JVM process. The JVM then generates a SIGSEGV signal in the child process. The SIGSEGV signal causes Linux to create a system dump for the child process. The parent JVM processes and renames the system dump, as required, by...
```
the -Xdump options, and might add additional data into the dump file. The system dump for
the child process contains an exact copy of the memory areas used in the parent. The SDK
dump viewer can obtain information about the Java threads, classes, and heap from the
system dump. However, the dump viewer, and other system dump debuggers show only the
single native thread that was running in the child process.

IBM Java then looks at /proc/PID/maps and tries to append information to the core dump that wouldn't
otherwise be there (in some cases this is not possible because the VMA does not have read permission):
"The Linux operating system core dump might not contain all the information included in a core dump
produced by the JVM dump agents."

In general, it is recommended to get the output of /proc/${PID}/smaps at the same time as getting a
system dump if you will be interested in virtual memory.

-Xdump

In recent versions, the defaults option may be used to change the default directory where certain
artifacts are attempted to be written to

-Xdump:java:defaults:file=/var/dumps/javacore.%Y%m%d.%H%M%S.%pid.%seq.txt

Starting with Java 7.1 SR2, Java 7.0 SR8, and Java 6.1 SR8 FP2, the default directory may be specified
for all artifacts (http://www-01.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.lnx.70.doc/diag/tools/dumpagents_tokens.html):

-Xdump:directory=/var/dumps

Stack Traces of Large Object Allocations

The filter is the number of megabytes:

-Xdump:stack:events=allocation,filter=#5m

Example output in stderr:

JVMDUMP039I Processing dump event "allocation", detail "5242880 bytes, type java.util.concurrent.ConcurrentHashMap$HashEntry[]" at 2015/09/14 07:36:49 -
please wait.
Thread=main (00007F8830007C30) Status=Running
   at java/util/concurrent/ConcurrentHashMap$HashEntry.newArray(I)
   [Ljava/util/concurrent/ConcurrentHashMap$HashEntry;
(ConcurrentHashMap.java:311) (Compiled Code)
   ...


To get a core dump instead, for example:

```
- Xdump:system:events=allocation,filter=#20m,range=1..1,request=serial+exclusive+prepwalk
```

**Thrown Exceptions**

-Xdump may be used to execute agents when an exception is thrown, including from a particular method. For example:

```
- Xdump:system:events=throw,range=1..1,request=serial+exclusive+prepwalk,filter=java/lang/NullPointerException#com/ibm/SomeClass.foo
```

Starting with Java 8, exceptions may be further filtered by the exception message: [http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/dumpagents_msg_filter.html](http://www-01.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/dumpagents_msg_filter.html). For example, to trigger a javacore on a java/lang/VerifyError exception that contains the text string "wrong initializer"

```
- Xdump:java:events=throw,filter=java/lang/VerifyError,msg_filter=*wrong initializer*
```

**Tool Agent**

The tool agent may be used to execute arbitrary process commands. For example, to print /proc/meminfo on Linux when there is an OOM:

```
- Xdump:tool:events=systhrow,filter=java/lang/OutOfMemoryError,request=serial+exclusive+prepwalk,range=1..0,priority=999,exec="cat /proc/%pid/smaps > smaps.%Y%m%d.%H%M%S.%pid.%seq.txt; cat /proc/meminfo > meminfo.%Y%m%d.%H%M%S.%pid.%seq.txt"
```

**Caught Exceptions**

-Xdump may be used to execute agents when an exception is caught, including from a particular method. For example:

```
- Xdump:system:events=catch,request=exclusive,range=1..1,filter=*#Play.foo
```

Use the second number in the range option to control the maximum number of core dumps produced. In the above example, no more than 1 cores will be produced.

The system dump agent is often useful because the system dump can be loaded in a tool such as the Memory Analyzer Tool and various stack frame locals may be reviewed that may help understand the exception.
Until Java 7.1, enabling certain -Xtrace options may affect the performance of the entire JVM (see the -Xtrace section in the IBM Java chapter).

Forced Garbage Collections

Forced garbage collections (System.gc() or Runtime.gc()) can be investigated by printing stack traces whenever they're called using the generic JVM argument:

-Xtrace:trigger=method{java/lang/Runtime.gc,jstacktrace},print=mt

Output goes to native_stderr.log. There may be some performance overhead to this option so before running in production (see the -Xtrace section in the IBM Java chapter), so test the overhead in a test environment. Example output:

12:02:55.436*0x191de00 mt.2 > java/lang/Runtime.gc()V Native method, This = 1b24188
12:02:55.463 0x191de00 mt.18 - Instance method receiver: java/lang/Runtime@00002B8F6249AA70 arguments: ()
12:02:55.463 0x191de00j9trc_aux.0 - jstacktrace:
12:02:55.464 0x191de00j9trc_aux.1 - [1] java.lang.Runtime.gc (Native Method)
12:02:55.464 0x191de00j9trc_aux.1 - [2] java.lang.System.gc (System.java:278)
12:02:55.464 0x191de00j9trc_aux.1 - [3] Test.main (Test.java:3)

If you are on IBM Java >=6 and < 7.1, then you may instead use -Xdump:stack:events=fullgc

This will print a stack trace to stderr every time a full garbage collection occurs:

Thread=WebContainer : 263509055 (F6C04688) Status=Running
   at java/lang/Runtime.gc()V (Native Method)
However, it will also print a stack any time a full GC occurs for non-explicit reasons. You can simply look for any stacks that begin with Runtime.gc to figure out which ones are explicit.


Requesting Full GCs

If it is required to request full GCs, here are some options (assuming -Xdisableexplicitgc is not set):

1. MBean: MemoryMXBean.gc:
   https://docs.oracle.com/javase/7/docs/api/java/lang/management/MemoryMXBean.html#gc
2. Create a JSP/Servlet or other type of application that executes System.gc based on some HTTP request or other input
3. Use the Java Surgery tool with -command CollectGarbage (this one is probably the closest to jcmd, although see the caveats on the page):
4. Use -Xdump or -Xtrace to trigger on certain method invocations (hard to configure)
5. Request a heapdump - this will force a GC as part of taking the heapdump:
   https://publib.boulder.ibm.com/httpserv/cookbook/Troubleshooting-Troubleshooting_WebSphere_Application_Server-Troubleshooting_WAS_Traditional.html#Troubleshooting-Troubleshooting_WAS_Traditional-Request_Heap_Dump

Stack Traces of the Sources of Threads

The sources of threads may be tracked by dumping where those threads are instantiated, which is likely the code that will subsequently spawn those threads. For example, if there is a thread with the following stack:

3XMTREADINFO3 Java callstack:
4XESTACKTRACE at java/lang/Object.wait(Native Method)
4XESTACKTRACE at java/lang/Object.wait(Object.java:196(Compiled Code))
4XESTACKTRACE at java/lang/ref/ReferenceQueue.remove(ReferenceQueue.java:102(Compiled Code))
4XESTACKTRACE at sun/rmi/transport/DGCClent$EndpointEntry$RenewCleanThread.run(DGCClent.java :492(Compiled Code))
4XESTACKTRACE at java/lang/Thread.run(Thread.java:736(Compiled Code))

Then the thread class is sun/rmi/transport/DGCClent$EndpointEntry$RenewCleanThread. Next, construct an -Xtrace option which prints the stack trace of the constructor call to native_stderr.log. For example:

-Xtrace:print=mt,methods={sun/rmi/transport/DGCClent$EndpointEntry$RenewClean...
Thread.<init>*},trigger=method{sun/rmi/transport/DGCClient$EndpointEntry$RenewCleanThread.<init>*},jstacktrace}

As another example, for Timers with stacks such as:

3XMTREADINFO    "Thread-2572" ...  
3XMTREADINFO3    Java callstack: 
4XESTACKTRACE    at java/lang/Object.wait(Native Method)  
4XESTACKTRACE    at java/lang/Object.wait(Object.java:196(Compiled Code))  
4XESTACKTRACE    at java/util/Timer$TimerImpl.run(Timer.java:246(Compiled Code))  
5XESTACKTRACE    (entered lock: java/util/Timer$TimerImpl@0x0000000720737468, entry count: 1)

Add the following generic JVM argument:

-Xtrace:methods={java/util/Timer$TimerImpl.<init>*},trigger=method{java/util/Timer$TimerImpl.<init>*},jstacktrace}

Example output:

20:05:02.535*0x23b3f500              mt.0        >  
java/util/Timer$TimerImpl.<init>(Ljava/lang/String;Z)V Bytecode method, This = 20884938  
20:05:02.535 0x23b3f500              mt.18       - jstacktrace:  
20:05:02.535 0x23b3f500              mt.6        <  
Java callstack:
java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 4125660  
20:32:41.615 0x13d86800              mt.0        >  
java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 4125660  
20:32:41.615 0x13d86800              mt.18       - Instance method receiver:  
java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 4125660  
20:32:41.855 0x13d86800              mt.6        <  
com.ibm.TimerTestServlet.service (TimerTestServlet.java:22)...  
java/util/Timer$TimerImpl.<init>(Ljava/lang/String;Z)V Bytecode method

Enabling certain -Xtrace options may affect the performance of the entire JVM (see the -Xtrace section in the IBM Java chapter).

Network Tracing

To enable tracing for the SDK's java/net classes, you may use:

-Xtrace:methods={java/net/*},print=mt

This writes to native_stderr.log. For example:

20:32:41.615 0x13d86800              mt.0        >  
java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 4125660  
20:32:41.615 0x13d86800              mt.18       - Instance method receiver:  
java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 4125660  
20:32:41.855 0x13d86800              mt.6        <  
java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method
Equivalently, the trace may be sent to files. For example:

-Xtrace:methods={java/net/*},output={jvmtrace#.trc,100M,10},maximal=mt

Then on each jvmtrace*.trc file, run the trace formatter; for example:

$ java com.ibm.jvm.TraceFormat jvmtrace0.trc

In the following example, we can see the first call doesn't find the host name in the cache, then puts it in the cache:

```
21:07:36.564789000  0x0000000013c6ba00 mt.0                Entry
>java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 0x435f2b0
21:07:36.564790000  0x0000000013c6ba00 mt.18               Event
Instance method receiver: java/net/Inet4Address@0x435f2b0 arguments: ()
...
```

```
21:07:36.783388000  0x0000000013c6ba00 mt.3                Entry
>java/net/InetAddress.cacheAddress(Ljava/lang/String;Ljava/lang/Object;Z)V Bytecode static method
...
```

```
21:07:36.783425000  0x0000000013c6ba00 mt.0                Entry
>s = 0x1f0e930
21:07:36.783428000  0x0000000013c6ba00 mt.18               Event
Instance method receiver: java/net/InetAddress$Cache@0x1f0e930 arguments: (java/lang/String@00000000043A9D60...000043AA260)
...
```

```
21:07:36.783656000  0x0000000013c6ba00 mt.6                Exit
<br/java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method
```

In the next call, there is no put; therefore, it found it in the cache:

```
21:07:41.373200000  0x0000000013c6ba00 mt.0                Entry
>java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method, This = 0x43ba250
21:07:41.373201000  0x0000000013c6ba00 mt.18               Event
Instance method receiver: java/net/Inet4Address@0x43ba250 arguments: ()
...
```

```
21:07:41.493092000  0x0000000013c6ba00 mt.3                Entry
>java/net/InetAddress.getCachedAddress(Ljava/lang/String;)Ljava/lang/Object; Bytecode static method
...
```

```
21:07:41.493165000  0x0000000013c6ba00 mt.6                Exit
<br/java/net/InetAddress.getCanonicalHostName()Ljava/lang/String; Bytecode method
```

Enabling certain -Xtrace options may affect the performance of the entire JVM (see the -Xtrace section in the IBM Java chapter).

**Debugging File Leaks**

If core dump analysis does not discover the cause of file leaks (this may be particularly difficult on
Windows when a particular leaked file must be found, because the file descriptor identifiers in Java objects do not directly map to Windows HANDLE addresses), then IO trace points may be used. IO trace points differ by operating system, so you may start with all IO trace points (print=IO), but in this example we show Windows trace points:

-Xtrace:print=IO.100-105,trigger=tpnid{IO.103,jstacktrace}

This also adds jstacktrace on IO.103 which is a file open.

```
21:40:27.491 0x2479c200              IO.103      >
IO_CreateFileW(filename=C:\WAS\profiles\...\.metadata\.plugins\...\properties .version...
21:40:27.491 0x2479c200       j9trc_aux.0        - jstacktrace:
21:40:27.491 0x2479c200       j9trc_aux.1        - [1]
java.io.FileOutputStream.open (Native Method)
21:40:27.491 0x2479c200       j9trc_aux.1        - [2]
java.io.FileOutputStream.<init> (FileOutputStream.java:233) (Compiled Code)
21:40:27.491 0x2479c200       j9trc_aux.1        - [3]
java.io.FileOutputStream.<init> (FileOutputStream.java:183) (Compiled Code)
21:40:27.491 0x2479c200       j9trc_aux.1        - [4]
org.eclipse.core.internal.localstore.BucketTree.saveVersion
(BucketTree.java:145)
21:40:27.491 0x2479c200       j9trc_aux.1        - [5]
org.eclipse.core.internal.localstore.BucketTree.close (BucketTree.java:80)
21:40:27.491 0x2479c200       j9trc_aux.1        - [6]
org.eclipse.core.internal.properties.PropertyManager2.shutdown
(PropertyManager2.java:169)
21:40:27.491 0x2479c200       j9trc_aux.1        - [7]
org.eclipse.core.internal.resources.Workspace.shutdown (Workspace.java:1829)
21:40:27.491 0x2479c200       j9trc_aux.1        - [8]
org.eclipse.core.internal.resources.Workspace.close (Workspace.java:369)
21:40:27.491 0x2479c200       j9trc_aux.1        - [9]
org.eclipse.core.resources.ResourcesPlugin.shutdown
(ResourcesPlugin.java:344)
21:40:27.491 0x2479c200       j9trc_aux.1        - [10]
org.eclipse.core.internal.compatibility.PluginActivator.stop
(PluginActivator.java:46)
21:40:27.491 0x2479c200       j9trc_aux.1        - [11]
org.eclipse.osgi.framework.internal.core.BundleContextImpl$2.run
(BundleContextImpl.java:843)
21:40:27.491 0x2479c200       j9trc_aux.1        - [12]
java.security.AccessController.doPrivileged (AccessController.java:341)
(Compiled Code)
21:40:27.491 0x2479c200       j9trc_aux.1        - [13]
org.eclipse.osgi.framework.internal.core.BundleContextImpl.stop
(BundleContextImpl.java:836)
21:40:27.491 0x2479c200       j9trc_aux.1        - [14]
org.eclipse.osgi.framework.internal.core.BundleHost.stopWorker
(BundleHost.java:501)
21:40:27.491 0x2479c200       j9trc_aux.1        - [15]
org.eclipse.osgi.framework.internal.core.AbstractBundle.suspend
(AbstractBundle.java:550)
21:40:27.491 0x2479c200       j9trc_aux.1        - [16]
org.eclipse.osgi.framework.internal.core.Framework.suspendBundle
(Framework.java:1101)
```
In the above example, an open returns the file handle 5072 (the result in the method exit of CreateFileW), and that is immediately followed by a CloseHandle on 5072, which succeeds. Finding an open without a close will discover the leak.

**Tracing Profiler**


**jdmpview/IDDE Commands**

Dump Java stack memory:

```
> xx 0xf1afdc10,10,4
0xf1afdc10 : 4C6DB928 00000000 E1A88FD0 00000000 [ (.mL............]
```

In the above example, 4C6DB928 was a Java object.

**-Xcheck**

-Xcheck:memory may be used to investigate native memory issues within the JVM itself:

Snap Traces

Snap traces contain tracepoint data held in JVM trace buffers (-Xtrace). Think of a snap trace as a black box flight recorder. An example file name is Snap.20140930.025436.9920.0004.trc. To process a snap trace, use the same Java version that produced the file to run the trace formatter on the snap file (http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/tools/trace_formatter.html). For example:

```
$ java com.ibm.jvm.TraceFormat Snap.20140930.025436.9920.0004.trc
```

If you are formatting a snap dump from another JVM, download their *.dat files ($WAS/java/jre/lib/) to a local directory and then use:

```
java com.ibm.jvm.format.TraceFormat Snap...trc.gz -datdir $DIR
```

On z/OS, FTP the .dat files down in ASC.

The TraceFormatter will produce an output file with the same name plus a .fmt suffix which is a human readable output of the snap file. For example:

```
09:54:40.168724484 *0x0000000002d2e4e00 j9prt.527           Exception  *
j9vmem_reserve_memory (failure)
09:54:40.168736220 0x0000000002d2e4e00 j9prt.468           Exception  *
allocate_memory32 failed to allocate byteAmount=8388608
callSite=segment.c:211
09:54:40.168740272 0x0000000002d2e4e00 j9prt.1049          Exception  *
vmem
allocate normal region failed. Callsite = segment.c:211, regionSize = 8388608
09:54:40.16874813 0x0000000002d2e4e00 j9prt.1045          Exception  *
memory32 allocate returned null pointer. Callsite = segment.c:211, byteAmount = 32832
09:54:40.168747956 0x0000000002d2e4e00 j9vm.199          Exception  *
Failed to allocate memory for segment in list 0x19fa50 (size=32768
type=0x10000)
09:54:40.168768913 0x0000000002d2e4e00 j9vm.201          Exit
<allocateMemorySegmentInList result=0x0
09:54:40.16876807 0x0000000002d2e4e00 j9vm.94          Exception  *
Unable to allocate 1912 bytes for RAM class. Throw OutOfMemoryError
```

Excessive Direct Byte Buffers

In addition to the section on excessive direct byte buffers in the general Troubleshooting Java chapter, IBM Java offers additional potential mitigations to excessive DBBs:

1. Use -Xgc:maxScavengeBeforeGlobal=N to force System.gc()s after every N scavenges. This option may have performance implications.
2. Use a non-generational garbage collection policy such as -Xgcpolicy:optthruput or -Xgcpolicy:optavgpause. This option may have performance implications. As the article quoted above mentions, this may not completely solve the issue.

A recent IBM javacore shows how much native memory is currently in use by DirectByteBuffers. For example:

```
5MEMUSER allocations | | | +--Direct Byte Buffers: 1,865,530,448 bytes / 150746
```
Direct byte buffer allocations and frees may be tracked with the following -Xtrace (http://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.diagnostics.60/diag/problem_determination/win_mem_trace_dbb.html?cp=SSYKE2_6.0.0%2F3-1-4-15-0&lang=en):
-Xtrace:print=j9jcl.335-338

OutOfMemoryError

Starting with IBM Java 6.0.1, a system dump is produced on the first OutOfMemoryError in addition to the previous artifacts (PHD, javacore, snap): http://www-01.ibm.com/support/docview.wss?uid=swg21584396

Native OutOfMemoryErrors on 64-bit

There are three broad types of native OutOfMemoryErrors on 64-bit:

1. With compressed references, insufficient virtual address space below 4GB for native classes, threads, and monitors.
2. Ulimit exhaustion on certain operating systems such as Linux and AIX.
3. If malloc or mmap return NULL or fail for any reason (e.g. physical memory and swap are exhausted). For example, on Linux, with overcommit_memory=2, if the amount of committed bytes (Committed_AS in /proc/meminfo) will exceed (Swap + (RAM*overcommit_ratio)) || (Swap + overcommit_kbytes).

When using IBM Java in 64-bit mode and with a maximum heap size less than 25GB, then Compressed References technology (-Xcompressedrefs) is enabled by default (defaults may be different on older versions of Java on some operating systems): http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/understanding/mm_compressed_references.html

This option will "decrease the size of Java objects and make more effective use of the available space. The result is less frequent garbage collection and improved memory cache utilization."
(http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/user/garbage_compressed_refs.html)

There are important implications to compressed references related to native OutOfMemoryErrors:

When you are using compressed references, the following structures are allocated in the lowest 4 GB of the address space: Classes, Threads, Monitors. Additionally, the operating system and native libraries use some of this address space. Small Java heaps are also allocated in the lowest 4 GB of the address space. Larger Java heaps are allocated higher in the address space.

Native memory OutOfMemoryError exceptions might occur when using compressed references if the lowest 4 GB of address space becomes full, particularly when loading classes, starting threads, or using monitors. You can often resolve these errors with a larger -Xmx option to put the Java heap higher in the address space.

A command-line option can be used with -Xcompressedrefs to allocate the heap you specify with the -Xmx option, in a memory range of your choice. This option is
-Xgc:preferredHeapBase=<address>, where <address> is the base memory address for the heap. In the following example, the heap is located at the 4GB mark, leaving the lowest 4GB of address space for use by other processes. -Xgc:preferredHeapBase=0x100000000

http://www.ibm.com/support/knowledgecenter/SSYKE2_8.0.0/com.ibm.java.lnx.80.doc/diag/understanding/mm_compressed_references.html

The first key point is that some maximum heap sizes below 4GB may cause the Java heap to be placed in the 0-4GB address space range (when possible). Compressed references technology works by compressing and decompressing pointers at runtime using bit shift arithmetic (ftp://public.dhe.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf). However, if the Java heap can be fit under 4GB, then these extra instructions are not required. In one benchmark, when the Java heap moved above the 0-4GB range, there was a relative throughput decrease of ~2.5% (ftp://public.dhe.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf). Note that this 2.5% effect was not under ceteris paribus conditions because the heap size was increased rather than using -Xgc:preferredHeapBase. The purpose of using -Xgc:preferredHeapBase (or alternatively, increasing the maximum heap size) is that you are forcing the JVM to take this performance hit in order to give more space to the native class, thread, and monitor data structures to avoid Native OutOfMemoryErrors (NOOMs).

The second key point is that native class, thread, and monitor data structures must all be allocated below 4GB when using compressed references. The operating system and other native allocations may further limit the available space under 4GB, so if you continue to get native OutOfMemoryErrors even with the Java heap allocated above the 0-4GB range, then you must address the number and size of the class, thread, and monitor data structures. In many cases, this is caused by a class, classloader, or thread leak which you can investigate with various tools, but it's easiest to start off by analyzing the javacore from the NOOM. If there are no leaks, then there may be other ways to reduce these data structures such as reducing reflection inflation, using shared classes, etc (see http://www-01.ibm.com/support/docview.wss?uid=swg27039764&aid=1).

One option to avoid these problems and NOOMs is to disable compressed references entirely; however, some benchmarks show a 10-20% relative throughput decrease when doing so: "Analysis shows that a 64-bit application without CR yields only 80-85% of 32-bit throughput but with CR yields 90-95%. Depending on application requirements, CR can improve performance up to 20% over standard 64-bit." (ftp://public.dhe.ibm.com/software/webserver/appserv/was/WAS_V7_64-bit_performance.pdf). You may be able to recover some of this drop by increasing L2/L3 processor cache sizes or efficiency (using processor sets). Disabling compressed references will also dramatically increase Java heap usage by up to 70% (because the pointers are doubled, the same Java object reference takes more of the Java heap).

For more information, see http://www-01.ibm.com/support/docview.wss?uid=swg21660890

Common causes of exhaustion below 4GB even if the heap is above:

1. Too many classes, classloaders, threads, or monitors.
2. Too many other, non-Class/Thread/Monitor allocations going below 4GB. Starting with Java 6.0.1 SR8 FP3 and Java 7 SR8 FP10, consider reserving more of this space for Classes/Threads/Monitors with -Xmcrs=MB. For older releases, an equivalent but undocumented and unsupported option is -Xgc:suballocatorInitialSize=#MB.

http://www-
3. On Windows, its default allocation strategy fills up the virtual memory below 4GB, which is not necessary. Set HKLM\System\CurrentControlSet\Control\Session Manager\Memory Management\AllocationPreference to (REG_DWORD)=0x10000 (http://www-01.ibm.com/support/knowledgecenter/SSYKE2_7.0.0/com.ibm.java.win.70.doc/diag/understanding/mm_compressed_references.html?lang=pl)

In IBM Java <= 5 for Linux, the IBM_MALLOCTRACE option is available which calls glibc's mtrace: http://www.gnu.org/software/libc/manual/html_node/Tracing-malloc.html. Starting with IBM Java 6, this option was changed to function equivalent to -Xcheck:memory:all instead of calling mtrace: http://www-01.ibm.com/support/knowledgecenter/SSYKE2_6.0.0/com.ibm.java.doc.diagnostics.60/diag/appendixes/env_var/env_jvm.html.

Useful jdmpview and IDDE:

- info thread * - In recent versions, includes detailed native thread information
- info mmap -verbose - On some operating systems such as Linux, includes detailed information available in /proc

Native Stack Size (-Xss)

Due to padding, alignment, and other operating system requirements, the actual native thread stack size may be larger than that specified by -Xss.

Known Crashes
/org/eclipse/swt/internal/cairo/Cairo._cairo_fill(Native Method) =>
Debug Mode

In one case, removing the options "-Xdebug -Xnoagent" improved debug performance by 300%. In general, the only required arguments are "-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=7777"

Troubleshooting Oracle Java

jmap

jmap is an unsupported memory diagnostics tool shipped with the JDK: "This utility is unsupported and may or may not be available in future versions of the JDK." (http://docs.oracle.com/javase/7/docs/technotes/tools/share/jmap.html).

histo

The `histo` option may be used to print a histogram of Java objects by class, including number of instances and number of bytes. The `live` option only counts reachable objects, although it does force a Full GC. Example:

$ jmap -histo 15078
num #instances #bytes class name
----------------------------------------------
1:  399  4747704  [I
2: 1565  151240  [C
3:  450   51456  java.lang.Class
4:  194   48144  [B
5: 1229  29496  java.lang.String ...

jinfo

jinfo is an unsupported tool shipped with the JDK which prints Java configuration of a live Java process or from a core dump: http://docs.oracle.com/javase/7/docs/technotes/tools/share/jinfo.html

Thread Dump

Oracle Java can produce a thread dump which is printed to stdout and details the activity of each thread. For example:

2015-03-13 07:58:11  
Full thread dump Java HotSpot(TM) 64-Bit Server VM (23.25-b01 mixed mode):

"pool-1-thread-8402" prio=3 tid=0x00000000010956f000 nid=0x3c0f waiting on condition [0xffffffff7868fe000]
  java.lang.Thread.State: TIMED_WAITING (parking)
    at sun.misc.Unsafe.park(Native Method)
    - parking to wait for <0xfffffffff90f54a0> (a
     java.util.concurrent.SynchronousQueue$TransferStack)
at java.util.concurrent.locks.LockSupport.parkNanos(LockSupport.java:226)
at java.util.concurrent.SynchronousQueue$TransferStack.awaitFulfill(SynchronousQueue.java:460)
at java.util.concurrent.SynchronousQueue$TransferStack.transfer(SynchronousQueue.java:359)
at java.util.concurrent.SynchronousQueue.poll(SynchronousQueue.java:942)
at java.util.concurrent.ThreadPoolExecutor.getTask(ThreadPoolExecutor.java:1068)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1130)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
at java.lang.Thread.run(Thread.java:724)...

The "nid" is the hexadecimal LWP ID: http://www-01.ibm.com/support/docview.wss?uid=swg21162381
The output of `prstat -p` shows the LWP in decimal form.

**Request Thread Dump**

1. On POSIX operating systems, by default, the command `kill -3 ${PID}` will request a thread dump.

**HPROF Heapdumps**

An HPROF heapdump contains the full Java heap object graph as well as Java object memory contents (for example, Strings, primitives, etc.). This is used for investigating OutOfMemoryErrors, tuning Java heap usage, etc.

By default, when a Java memory request cannot be fulfilled, an OutOfMemoryError is thrown, but an HPROF dump is not produced. Use `-XX:+HeapDumpOnOutOfMemoryError` to produce an HPROF dump in this condition: http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html. Starting with Java 6, by default (-XX:+UseGCOverheadLimit), when garbage collection is more than 98% of the processing time of the process (-XX:GCTimeLimit=98) and less than 2% of the heap is being recovered (-XX:GCHepFreeLimit=2), an OutOfMemoryError is thrown with the details "GC overhead limit exceeded" (an HPROF dump is only produced with `-XX:+HeapDumpOnOutOfMemoryError`): http://www.oracle.com/technetwork/java/javase/gc-tuning-6-140523.html. HeapDumpOnOutOfMemoryError only produces a dump on the first OOM: http://bugs.java.com/bugdatabase/view_bug.do?bug_id=6280629

-XX:HeapDumpPath may be used to control where the dumps are written to: http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html

Some HPROF options may be changed while the JVM is running using MBeans (e.g. jconsole).

-XX:OnOutOfMemoryError may be used to execute an operating system command on an OOM: http://www.oracle.com/technetwork/java/javase/tech/vmoptions-jsp-140102.html
On recent versions of HotSpot Java, an HPROF heapdump also includes thread information which describes which Java objects are stack frame locals on which stacks (for example, you can see the actual SQL string for an executing database query). Available with Java 6 Update >= 14 and Java 7: https://bugs.eclipse.org/bugs/show_bug.cgi?id=268458

To analyze heapdumps, see the IBM Memory Analyzer Tool chapter.

**Generating HPROF heapdumps**

Additional methods of requesting heap dumps are documented in the Troubleshooting Operating Systems.

1. Automatically produced on OOM with -XX:+HeapDumpOnOutOfMemoryError
   When a heapdump is produced due to this option, output such as the following will appear in stdout:
   ```
   java.lang.OutOfMemoryError: GC overhead limit exceeded
   Dumping heap to java_pid28537.hprof ...
   ```
2. Ctrl+Break or kill -3 with -XX:+HeapDumpOnCtrlBreak
3. Pass the PID to the jmap tool. Note: The jmap tool is unsupported: "This utility is unsupported and may or may not be available in future versions of the JDK."
   (http://docs.oracle.com/javase/7/docs/technotes/tools/share/jmap.html).
   ```
   $ jmap -dump:format=b,file=heap.hprof ${PID}
   ```
4. Produce an operating system core dump (see the Troubleshooting Operating Systems chapter) and then extract the HPROF heapdump from the core dump:
   ```
   $ jmap -dump:format=b,file=heap.hprof ${PATH_TO_JAVA} ${PATH_TO_CORE}
   ```
5. Use -XX:OnOutOfMemoryError (see below)
6. Jconsole with HotSpotDiagnostic Mbean dumpHeap
7. From within MAT: File > Acquire Heap Dump

**Use DTrace to Produce Stacks Calling certain Methods**

The following DTrace script prints stacks when System.gc is called. The JVM must be started with -XX:+ExtendedDTraceProbes

Attach the DTrace script to a running PID: /usr/sbin/dtrace -qs methodtrace.d -p ${PID}

It is often the case that the overhead of such a DTrace script is very high and may be inappropriate for a production environment.

```d
#pragma D option bufsize=128m
dtrace:::BEGIN
{
   /* Java level tracing */
   traceJava = 1;       /* 1 = enable, 0 = disable */
   classFilter = "java/lang/System";    /* e.g. "java/util" */
   methodFilter = "gc";    /* e.g. "get" */

   /* Initialise the per-thread indentation variable */
   self->indent = 0;
}
```
hotspot$target:::method-entry
/
  traceJava
  && (classFilter == "" || strstr(copyinstr(arg1, arg2), classFilter) != NULL)
  && (methodFilter == "" || strstr(copyinstr(arg3, arg4), methodFilter) != NULL)
/
{
  self->indent += 2;

  wt = walltimestamp;
  printf("%Y.%09d: %d/%d:%*s-> %s.%s%s\n", 
         wt, 
         wt % 1000000000, 
         pid, 
         tid, 
         self->indent, 
         "", 
         copyinstr(arg1, arg2), 
         copyinstr(arg3, arg4), 
         copyinstr(arg5, arg6));
  jstack(500, 8192);
}

Example output:

2018 Jan 10 14:03:24.280004000: 18021/2:
  -> java/lang/System.gc()V
  libjvm.so`__1cNSharedRuntimeTdtrace_method_entry6FpnKJavaThread_pnNmethodOopD
  esc__i_+0x1ac
    java/lang/System.c
    SystemDemo.ai
    0xfb0021c
  libjvm.so`__1cJJavaCallsLcall_helper6FpnJJavaValue_pnMmethodHandle_pnRJavaCal
  lArguments_pnGThread__v_+0x318
    libjvm.so`jni_CallStaticVoidMethod+0x67c
    libjli.so`JavaMain+0x740
    libc.so.1`_lwp_start

Use -XX:OnOutOfMemoryError to Spawn jmap

#!/bin/sh
# Usage:
#  1. Create oom.sh with the contents of this script
#  2. Change paths in the "Variables" section if needed
#  3. chmod a+x oom.sh
#  4. Run java with the following argument, replacing $PATH with path to
#     oom.sh
#     -XX:OnOutOfMemoryError="/PATH/oom.sh %p"

# Variables
LOCKFILE=/tmp/oomlock
OUT=/tmp/oomout.txt
NOW=`date "%Y%m%d_%H%M%S"`
CURDIR=`pwd`
JAVA_HOME=/opt/IBM/WebSphere/AppServer/java/

# Execution
echo "OOM handler script started for PID $1 at $NOW in $CURDIR" >> $OUT
if [ ! -f $LOCKFILE ]; then
touch $LOCKFILE >> $OUT 2>&1
NOW=`date "%Y%m%d_%H%M%S"`
echo "OOM handler requested hprof at $NOW" >> $OUT
FILENAME="heap_${NOW}_${1}.hprof"
$JAVA_HOME/bin/jmap -F -dump:format=b,file=$FILENAME $1 >> $OUT 2>&1
# /usr/bin/gcore -F -o core_$NOW.dmp $1 >> $OUT 2>&1
CODE=$?
echo "OOM handler returned with $CODE at $NOW" >> $OUT
rm -f $LOCKFILE >> $OUT 2>&1
fi
NOW=`date "%Y%m%d_%H%M%S"`
echo "OOM handler finished at $NOW" >> $OUT

Use -XX:OnOutOfMemoryError to Spawn gcore

#!/bin/sh
# Usage:
#  1. Create oom.sh with the contents of this script
#  2. Change paths in the "Variables" section if needed
#  3. chmod a+x oom.sh
#  4. Run java with the following argument, replacing $PATH with path to
#     oom.sh
#   -XX:OnOutOfMemoryError="/PATH/oom.sh %p"
#  5. After an OOM occurs, check /tmp/oomout.txt for output of the command
#  6. Run `jmap -dump:format=b,file=heap.hprof ${PATH_TO_JAVA} ${CORE}`
#   
# Notes:
#  OnOutOfMemoryError runs /usr/bin/sh -c $CMD synchronously, during which it
#  appears it has a lock that prevents jmap to attach. Tried -F, but that
#  generated an infinite loop and ran into other issues, so running gcore
#  instead.

# Variables
LOCKFILE=/tmp/oomlock
OUT=/tmp/oomout.txt
NOW=`date "%Y%m%d_%H%M%S"`
CURDIR=`pwd`
GCORE_PATH=/usr/bin/gcore

# Execution
echo "OOM handler script started for PID $1 at $NOW in $CURDIR" >> $OUT
if [ ! -f $LOCKFILE ]; then
touch $LOCKFILE >> $OUT 2>&1
NOW=`date "%Y%m%d_%H%M%S"`
echo "OOM handler requested hprof at $NOW" >> $OUT
# $JAVA_HOME/bin/jmap -dump:format=b,file=heap_$1.hprof $1 >> $OUT 2>&1
$GCORE_PATH -F -o core_$NOW.dmp $1 >> $OUT 2>&1

CODE=$?
    echo "OOM handler returned with $CODE at $NOW" >> $OUT
    rm -f $LOCKFILE >> $OUT 2>&1
fi
NOW=`date "+%Y%m%d %H%M%S"`
    echo "OOM handler finished at $NOW" >> $OUT

Code to Request Diagnostics from within the JVM

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.lang.management.ManagementFactory;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.concurrent.atomic.AtomicInteger;

public class Play
{
    public static void main(String... args) throws Throwable
    {
        System.out.println("Requesting core...");
        tryGenerateCore();
    }

    public static void tryGenerateCore()
    {
        try
        {
            String requestedFileName = generateCore();
            if (requestedFileName != null)
            {
                System.out.println("Started writing core dump to " +
                        requestedFileName);
            }
        }
        catch (Throwable t)
        {
            System.out.println("Error generating core: " +
                        t.getLocalizedMessage());
            t.printStackTrace();
        }
    }

    private final static boolean ENABLE_REQUESTING_COREDUMPS =
            Boolean.getBoolean("ENABLE_REQUESTING_COREDUMPS");
    private final static SimpleDateFormat DIAG_NAME_FORMAT = new
            SimpleDateFormat("yyyyMMdd.HHmmss");
    private final static String CORE_PROGRAM_PATH =
            System.getProperty("CORE_PROGRAM_PATH", "/usr/bin/gcore");
    private final static int MAX_CORE_DUMPS =
            Integer.getInteger("MAX_CORE_DUMPS", 1);
    private static final AtomicInteger coreDumpsTaken = new AtomicInteger();
    private static int coreDumpsRequested;
```
/**
 * Disabled by default. Enable with -DENABLE_REQUESTING_COREDUMPS=true
 * <p />
 * Request a non-destructive core dump in a separate thread by spawning
 * out
 * to the gcore command. gcore will attach to and pause the process, dump
 * all virtual memory (so the size will be about the size in ps VSZ) and
 * then the process should continue. Unlike an OOM or using jmap to
 * request
 * an HPROF dump, requesting a core does not request a Full GC. Jmap can
 * be
 * used to extract an HPROF heapdump from the core:
 * <p />
 * <code>$ jmap -dump:format=b,file=heap.hprof ${PATH_TO_java} $ \{CORE\} </code>
 * <p />
 * Whereas asking the JVM to generate a heapdump with jmap is a complex
 * operation because the JVM has to walk all the data structures, the
 * operating system generating a core is very simple: the OS just pauses
 * the
 * process and dumps out all of the virtual memory. The overhead of a
 * core
 * file is almost completely in writing the large amount of bytes to
 * disk.
 * <p />
 * There are some techniques to make this very fast. First, if there is
 * sufficient filecache in RAM (i.e. a large amount of free RAM), then
 * the
 * OS will write the core to RAM and then asynchronously write to disk,
 * thus
 * making the pause quite fast. However, this can have some performance
 * side
 * effects. An alternative way to do this is to mount a RAMdisk and write
 * the core to a RAMdisk.
 * <p />
 * Warning: ensure sufficient core, file and other ulimits. Also ensure
 * sufficient disk space in the current working directory.
 * @return null if -DMAX_CORE_DUMPS (default 1) has been reached or
 * -DENABLE_REQUESTING_COREDUMPS=false; otherwise, the requested
 * core file name.
 * @throws IOException
 * @throws InterruptedException
 */

public static synchronized String generateCore() throws IOException,
    InterruptedException {
    if (!ENABLE_REQUESTING_COREDUMPS || coreDumpsRequested++ >=
        MAX_CORE_DUMPS) { return null; }
    CoreDumpThread coreDumpThread = new CoreDumpThread();
    coreDumpThread.start();
    return coreDumpThread.getRequestedFileName();
}

public static int getPID() {

String name = ManagementFactory.getRuntimeMXBean().getName();
if (name != null)
{
    int x = name.indexOf('@');
    if (x != -1)
    {
        name = name.substring(0, x);
        return Integer.parseInt(name);
    }
}
throw new RuntimeException("Could not find PID");

static class CoreDumpThread extends Thread
{
    private final int pid;
    private final String requestedFileName;
    private Throwable error;

    public CoreDumpThread()
    {
        super("CoreDumpThread : " + coreDumpsTaken.get());
        // Writing the core can take a while, so we'll prefer to block the
        // JVM
        setDaemon(false);
        pid = getPID();
        requestedFileName = "core." + DIAG_NAME_FORMAT.format(new Date())
            + "." + pid + ".dmp";
    }

    @Override
    public void run()
    {
        try
        {
            ProcessBuilder processBuilder = new
            ProcessBuilder(CORE_PROGRAM_PATH, "-o", requestedFileName, "" + pid);
            processBuilder.redirectErrorStream(true);
            Process process = processBuilder.start();
            BufferedReader br = new BufferedReader(new
            InputStreamReader(process.getInputStream()));
            String line;
            StringBuilder sb = new StringBuilder();
            while ((line = br.readLine()) != null)
            {
                sb.append(line);
            }
            int exitValue = process.waitFor();
            if (exitValue == 0)
            {
                coreDumpsTaken.incrementAndGet();
            }
            else
            {
                // Handle error
            }
        }
    }
}
System.out.println("Error requesting core. Exit value " + exitValue + ". Output " + sb.toString());
}

try {
    String result = Shell.runCommand(command, outStream, errStream);
    requestedFileName = result;
} catch (IOException e) {
    System.out.println("IOException: " + e.getMessage());
    errStream.println("IOException: " + e.getMessage());
}

public String getRequestedFileName()
{
    return requestedFileName;
}

public Throwable getError()
{
    return error;
}

Troubleshooting WebSphere Application Server

Additionally, see the chapter for Troubleshooting WAS Traditional or Troubleshooting WAS Liberty.

Notes

Any class packages that start with com.ibm.websphere are public. Those that start with com.ibm.ws are internal.

Increasing Resiliency for IBM WebSphere Application Server Deployments

The top practices that we have observed in customer situations which cause problems are (http://www.redbooks.ibm.com/redpapers/pdfs/redp5033.pdf):

1. No test environment is equal to the production environment
2. Communication breakdown
3. No plan for education
4. No load or stress testing
5. Not managing the entire application lifecycle
6. No capacity or scalability plan
7. No production traffic diagram
8. Changes are put directly into production
9. No migration plan
10. No record of changes
11. No current architecture plan
Troubleshooting WAS Traditional

WAS and Java

When upgrading WAS fixpacks, the Java fixpack should also be upgraded: https://www-304.ibm.com/support/docview.wss?uid=swg27005002

The Java SDK can be upgraded to any version available for that WAS major version. For example, if you are on WAS 6.1.0.21 but you want to run the 6.1.0.29 Java SDK, that is supported, although this is obviously a less tested configuration. You cannot use a WAS 7 SDK on WAS 6.1, for example.

PID File

It is possible to automate finding the process ID of particular application server through scripts. Each application server writes a file named ${SERVER}.pid into its log folder on startup. For example, on POSIX systems:

```
$ someScript.sh `cat /opt/IBM/WebSphere/AppServer/profiles/profile1/logs/server1/*.pid`
```

Stopping Servers

There are four ways to stop a WAS server (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/urn_rsvr.html?lang=en):

1. WAS Stop: Quiesce the server so that no new work is allowed in, allow existing work tracked by WAS to finish, then gracefully stop all applications, shutdown WAS components, and attempt to gracefully exit the Java process. By default, WAS will wait up to 3 minutes for the quiesce to complete. This can be changed with com.ibm.ejs.sm.server.quiesceTimeout: http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/xrun_jvm.html?cp=SSAW57_8.5.5%2F3-18-6-481&lang=en
2. WAS Immediate Stop: This is the same as a WAS Stop, except that it does not wait for existing work to finish. Based on tests on V9, ImmediateStop still waits for in-flight requests to finish just like the Stop function.
3. WAS Terminate: Unlike the stop and immediate stop methods, this method does not attempt to gracefully exit the Java process, but instead uses operating system commands to kill the process.
4. Operating system signal: Depending on the type of signal, either the process will end without any handling within WAS (destructive signal, e.g. SIGKILL) or as a WAS Immediate Stop (e.g. SIGTERM). WAS accomplishes the latter through a shutdown hook (http://docs.oracle.com/javase/7/docs/api/java/lang/Runtime.html#addShutdownHook%28java.lang.Thread%29).

It is recommended to first try a WAS Stop, wait 3 minutes, then try a WAS Immediate Stop, and finally try a WAS Terminate.

In the case of a WAS Terminate or a destructive operating system signal, the following are examples of some possible effects:
1. Transaction log: If an application uses transactions and the process ended during an in-flight transaction, the transaction log may need to be processed.
2. OSGi cache: If the process ended during OSGi activity, the OSGi cache may need to be reset with osgiCfgInit and clearClassCache.
3. IBM Java shared class cache: If the process ended during IBM Java shared class cache activity, the cache may need to be reset with Java commands.
4. HTTP sessions: If HTTP sessions are configured for distribution or persistence, some sessions may not have been committed and their states will be lost.

**Request Thread Dump**

Additional methods of requesting thread dumps are documented in the Troubleshooting Java chapters.

1. On IBM Java, use `wsadmin -lang jython` and run the following command, replacing server1 with the server name:
   
   AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "dumpThreads")

2. On Windows, use the `windows_hang.py` script which essentially does the same as #1 with much more flexibility: [http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg21111364](http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg21111364)

3. On IBM Java and WAS >= 8, use the administrative console, Java dumps and cores pane:

4. On Windows, generate an attached start server script with `startServer.bat -script`, start the server using the generated script, and now since you have an attached console, you can type Ctrl+Break to request a thread dump.

**JVM.dumpThreads**

The dumpThreads functionality is different depending on the operating system:

- POSIX (AIX, Linux, Solaris, etc.): kill(pid, SIGQUIT)
• Windows: raise(SIGBREAK)
• z/OS: In recent versions, produces a javacore, heapdump, and SYSTDUMP by default

For any customers that have changed the behavior of the JVM (-Xdump) in how it responds to SIGQUIT/SIGBREAK (i.e. kill -3), then dumpThreads will respond accordingly (unless running z/OS, in which case use wsadmin_dumpthreads* properties).

Request Heap Dump

Additional methods of requesting heap dumps are documented in the Troubleshooting Java chapters.

1. On IBM Java, use `wsadmin -lang jython` and run the following command, replacing server1 with the server name:
   AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "generateHeapDump")

2. On IBM Java and WAS >= 8, use the administrative console, Java dumps and cores pane:

Request System Dump

Additional methods of requesting system dumps are documented in the Troubleshooting Operating Systems and Troubleshooting Java chapters.

1. On IBM Java and WAS >= 8, use `wsadmin -lang jython` and run the following command, replacing server1 with the server name:
   AdminControl.invoke(AdminControl.completeObjectName("type=JVM,process=server1,*"), "generateSystemDump")

2. On IBM Java and WAS >= 8, use the administrative console, Java dumps and cores pane:
ClassLoader Leaks

ClassLoader leaks become most evident when an application is restarted and its old classes are not available for garbage collection. This may induce longer garbage collection times, Java OutOfMemoryErrors, and native OutOfMemoryErrors.

- The IBM Extensions for Memory Analyzer in the IBM Memory Analyzer Tool provide two classloader leak detection queries.

Thread IDs in Logs

Before WAS 8.5, the "thread ID" printed in WAS logs (the hexadecimal number after the timestamp) comes from the java/util/logging/LogRecord.getThreadID method. This number was not in javacores, so there was no easy way to correlate javacores with log and trace messages. Moreover, this thread ID was different from java/lang/Thread.getID which might be printed in other components, and that thread ID also wasn't in javacores. There were some complex techniques of correlating IDs: http://www-304.ibm.com/support/docview.wss?uid=swg21418557

WAS 8.5 has changed the ID printed in logs to the value from java/lang/Thread.getID. This can be changed back to the previous behavior using com.ibm.websphere.logging.useJULThreadID=true. See: http://www14.software.ibm.com/webapp/wsbroker/redirect?version=phil&product=was-nd-mp&topic=rtrb_readmsglogs. This was also backported to WAS 8.0.0.4 (PM60913); however, it must be explicitly enabled in WAS 8.0.

With IBM Java 6 SR12 (WAS 7.0.0.27), IBM Java 626 SR4 (WAS 8.0.0.6, 8.5.0.2, 8.5.5.0), and IBM Java 7 SR3 (WAS 8.5.0.2, 8.5.5.0), javacores will have the value of Thread.getID printed with each stack. Given the above change, this allows you to correlate WAS log messages with javacores (note the bold and underlined parts):

[8/22/12 10:00:05:049 PDT] 0000005b SystemOut O swat.ear: Calling
TrapIt.ear

TrapIt.ear is a free enterprise application which may be installed to watch for particular log messages and generate diagnostics such as thread dumps, as well as time-based triggers to do the same:
http://www-01.ibm.com/support/docview.wss?uid=swg21644180

High Availability Manager

JVM Panic

Under some conditions, the High Availability Manager will "panic," print some diagnostics, and then force the WAS process to stop itself. The symptoms of this will include:

1. A stack trace in RuntimeProviderImpl.panicJVM in SystemErr.log. For example:

```
[1/1/15 00:00:00:000 UTC] 00000001 SystemErr     R java.lang.Throwable
[1/1/15 00:00:00:000 UTC] 00000001 SystemErr     R     at java.lang.Thread.dumpStack(Thread.java:434)
[1/1/15 00:00:00:000 UTC] 00000001 SystemErr     R     at com.ibm.ws.hamanager.runtime.RuntimeProviderImpl.panicJVM(RuntimeProviderImpl.java:91)
[1/1/15 00:00:00:000 UTC] 00000001 SystemErr     R     at com.ibm.ws.hamanager.impl.HAGroupImpl.doIsAlive(HAGroupImpl.java:882)
```

2. A "Panic" line in SystemOut.log with a detailed description of the reason for the panic. For example:

```
Panic:component requested panic from isAlive
```
3. A stack trace in ServerImpl.emergencyShutdown in SystemOut.log. For example:

```
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  java.lang.RuntimeException: emergencyShutdown called:
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  at com.ibm.ws.runtime.component.ServerImpl.emergencyShutdown(ServerImpl.java:633)
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  at com.ibm.ws.hamanager.runtime.RuntimeProviderImpl.panicJVM(RuntimeProviderImpl.java:92)
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  at com.ibm.ws.hamanager.coordinator.impl.JVMControllerImpl.panicJVM(JVMControllerImpl.java:56)
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  at com.ibm.ws.hamanager.impl.HAGroupImpl.doIsAlive(HAGroupImpl.java:866)
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  at com.ibm.ws.hamanager.impl.HAGroupImpl$HAGroupUserCallback.doCallback(HAGroupImpl.java:1364)
[1/1/15 00:00:00:000 UTC] 00000001 SystemOut     O  at com.ibm.ws.hamanager.impl.Worker.run(Worker.java:64)
[1/1/15 00:00:00:000 UTC] 00000001 SibMessage    I   CWSIS1519E: Messaging engine ${ME} cannot obtain the lock on its data store, which ensures it has exclusive access to the data.
[1/1/15 00:00:00:000 UTC] 00000001 SibMessage    E   CWSID0046E: Messaging engine ${ME} detected an error and cannot continue to run in this server.
[1/1/15 00:00:00:000 UTC] 00000001 HAGroupImpl   I   HMGR0130I: The local member of group ...,WSAF_SIB_MESSAGING_ENGINE=...,type=WSAF_SIB has indicated that is it not alive. The JVM will be terminated.
```

One common cause of these panics is that the SIB messaging engine cannot communicate with its data store due to a database error. For example, messages such as the following precede the panic:

```
[1/1/15 00:00:00:000 UTC] 00000001 ConnectionEve A   J2CA0056I: The Connection Manager received a fatal connection error from the Resource Adapter for resource jdbc/sibdb. The exception is: com.ibm.db2.jcc.am.zn: [jcc] [t4][2030][11211][3.57.110] A communication error occurred during operations on the connection's underlying socket, socket input stream, or socket output stream. Error location: Reply.fill(). Message: Insufficient data. ERRORCODE=-4499, SQLSTATE=08001
[1/1/15 00:00:00:000 UTC] 00000001 SibMessage    I   CWSIS1519E: Messaging engine ${ME} cannot obtain the lock on its data store, which ensures it has exclusive access to the data.
[1/1/15 00:00:00:000 UTC] 00000001 SibMessage    E   CWSID0046E: Messaging engine ${ME} detected an error and cannot continue to run in this server.
[1/1/15 00:00:00:000 UTC] 00000001 HAGroupImpl   I   HMGR0130I: The local member of group ...,WSAF_SIB_MESSAGING_ENGINE=...,type=WSAF_SIB has indicated that is it not alive. The JVM will be terminated.
```

This is expected behavior and the database needs to be investigated or the data source configuration needs to be tuned: "Behavior when the data store connection is lost... default: The high availability manager stops the messaging engine and its hosting application server when the next core group service Is alive check takes place (the default value is 120 seconds)." (http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.nd.multiplatform.doc/ae/tjm_dsconnloss.html)

**Messaging**

**Users Seeing other Users' Data**

The call center is getting reports and screenshots from users who see not their own data but another user's data. Depending on the number of reports received it may necessitate shutting down the web site until the problem can be diagnosed and fixed. This is one of the most difficult problems troubleshoot and identify root cause. Try to recreate the problem in one of the application test environments. Hopefully it is easily recreateable, however, the symptom may not exhibit itself in a test environment.

**Strategy 1: Open a PMR with IBM Support**

It is imperative to open a PMR with IBM Support immediately and verify with the support specialist that there is no known issue for which a known APAR is available. If there is an APAR then download and install the APAR and restart the application server environment.

**Monitor**

Monitor reports from the user community and if reports continue to come in that users see other user's data then pursue one of the other strategies.

**Caveat**

Applying the APAR does not guarantee it will fix the problem if the issue resides within the application code itself. The APAR is only applicable if it is a bug in the WebSphere Application Server.

**Strategy 2: Application code review**

Review the application code and look for one of the following anti-patterns that may be causing users to see another user's data. In no particular order:

- Not clearing thread local variables. [note: was this a feature we added to WAS and no longer a problem since some version?]
- Storing data within the Servlet in an instance variable defined at the Servlet class.

**Monitor**

Fix the code, rebuild the application, redeploy and test the application. Once it passes the testing process deploy to production. Monitor reports from the user community.

**Caveat**

Because this type of application bug is so difficult to diagnose and resolve the application "fix" may not actually fix the problem. Because of there being multiple bugs there may be several iterations of of this strategy.

**DRS or HA Manager Errors**

Distributed Replication Server or HA Manager errors appear in the logs of either the DMgr or the nodeagents or application server logs themselves.
Strategy 1: Check version/fixpack level of DMgr and JVMs putting out errors

Sometimes a fixpack may be inadvertently missed on a particular JVM or node. Apply the fixpack, restart and see if that fixes the problem.

Monitor

Logs for recurrence of the error

Caveats

Sometimes applying a fixpack may negatively affect an application. Make sure to test all fixpacks before applying them in production.

Strategy 2: Application code is using DistributedMap class

An application that aggressively uses DistributedMap may negatively increase the amount of communication between the application servers. This typically exhibits itself when either the number of application server JVMs is increased or the thread pool for the application is increased inside the JVM. This also inhibits the ability to grow the environment as the user base grows. Therefore, reconsider the use of DistributedMap in an application particularly for high volume, business critical applications.

Monitor

Logs for recurrence of the error.

Caveats

Re-architecting/desigining and re-coding the application to eliminate the use of DistributedMap can take weeks to months depending on how extensively DistributedMap was used.

Application Works in some Nodes

The same EAR file is deployed to two nodes. It works on one node but not another.

Strategy 1: NoClassDefFoundError thrown


Analyze the data. Following things should be checked very carefully.

In the trace.log find the full "Local Classpath" for the application. Compare the working node to the non-working node. Very often the administrator is somehow not deploying the application correctly and puts in a classpath that is either missing items or rearranging the order which picks up a different version of the same jar file.
In the one case I worked today: Application worked on AIX/Windows but NoClassDefFoundError on Linux. It is a JSF application but not supposed to use JSF2. But the jsf2 lib was included in the EAR file. On the working nodes the non-JSF2 impl lib was preceding the JSF2 lib. On the non-working Linux node they were reversed and of course the JSF2 class had a dependency they must not have included.

**Troubleshooting WAS Liberty**

**Server Dumps**

The server dump command requests various types of status information of a running server:


```
$ cd {WLP}
$ server dump ${SERVER} # Dump general server information
```

The produced files will go to `{WLP}/usr/servers/${SERVER}`

**Request Thread Dump**

Additional methods of requesting thread dumps are documented in the Troubleshooting Java chapters.

```
$ cd {WLP}
$ server javadump ${SERVER}
```

**Request Heap Dump**

Additional methods of requesting heap dumps are documented in the Troubleshooting Java chapters.

```
$ cd {WLP}
$ server javadump ${SERVER} --include=heap
```

**Request System Dump**

Additional methods of requesting system dumps are documented in the Troubleshooting Operating Systems and Troubleshooting Java chapters.

```
$ cd {WLP}
$ server javadump ${SERVER} --include=system
```

**Client fails to Connect to Liberty Messaging Engine**

The Liberty logs show the application has started normally and bound to the correct ports. However, the client application is getting the error.

1. Telnet to the IP port# fails.
2. netstat -an |grep LIST on the Liberty server shows the port is bound to 127.0.0.1 (localhost).

com.ibm.ws.sib.jfapchannel.JFapConnectFailedException: CWSIJ0063E: A network connection to host name /192.168.2.234, port 9,126 cannot be established...

**Strategy:** Fix server.xml endpoints

Make sure the server.xml entries for the endpoints attribute host is set as:

```
host="*"
```

---

**Troubleshooting WebSphere MQ**

**Documentation**


**Basic Display Commands**

- dspmqinst lists the MQ installations on the machine
- dspmqver shows the MQ version and patch level
- dspmq lists queue managers on the local machine, and the status of each one

**Multiple Installations of MQ on the Same Machine**

Starting with MQ v7.1, it is possible to install multiple copies of MQ (of the same or different version) on a single machine. (Prior to this, MQ directory pathnames had been hard-coded so it was not possible to install more than one copy of MQ.) Each separate instance of MQ on a machine is referred to as an "installation," and you can choose where in the filesystem each installation should be based, subject to a few restrictions. All installations still share a common MQ data directory tree -- it is only the MQ binaries which are kept separate for different installations. On Unix-like systems, the `/etc/opt/mqm/mqinst.ini` command lists the currently-existing installations, and the directory path to each one. The dspmqinst command also lists the installations on a machine.

There is a command named setmqinst which can be used to set all appropriate environment variables to point to a particular installation, as a means of determining which of the multiple MQ installations on a machine will be referenced when you issue other MQ commands. For example, ".

`/opt/mqm/bin/setmqenv -s` on a Linux machine sets the MQ environment variables to refer to the copy of MQ that lives in `/opt/mqm`. If you are having problems with "command not found" errors or the like, you may need to issue the setmqenv command. Each queue manager is associated with a particular installation, so you may also need to issue setmqinst if you get errors saying that your queue manager is associated with a different installation.

**Log Files**

- There is a "high-level" errors directory at the top of the MQ tree, and each queue manager also has its own errors directory. The high-level errors directory has messages that do not pertain to a
specific queue manager. Note that the high-level MQ directory named "log" contains transaction logs, not error logs.

- Unix default locations: /var/mqm/errors and /var/mqm/qmgrs/<QM_NAME>/errors
- Windows prior to MQ v8.0: \Program Files\IBM\WebSphere MQ\errors and \Program Files\IBM\WebSphere MQ\qmgrs\<QM_NAME>\errors
- Windows v8.0: C:\ProgramData\IBM\MQ\errors and C:\ProgramData\IBM\MQ\qmgrs\<QM_NAME>\errors; note that the C:\ProgramData directory is typically a "hidden" directory
- Each "errors" directory always contains exactly 3 log files: AMQERR01.LOG, AMQERR02.LOG, and AMQERR03.LOG
- MQ automatically rolls the log files, so AMQERR01.LOG is always most recent
- Maximum size can be controlled via ErrorLogSize in the QMErrorLog stanza of qm.ini on Unix, or via MQ Explorer on Windows (queue manager Properties > Extended)
- Application event log on Windows also contains MQ events
- Location of error logs on all MQ platforms: http://www-01.ibm.com/support/docview.wss?uid=swg21172370

**Reason Codes and Error Messages**

- The mqrc command can decode a 4-digit MQ reason code, for example: mqrc 2035
- Understanding common MQ reason codes: http://www-01.ibm.com/support/docview.wss?uid=swg21167821
- Common MQ error messages (AMQxxxx codes) and most likely causes: http://www-1.ibm.com/support/docview.wss?uid=swg21265188
- Complete list of MQ 8.0 reason codes
- 2007 MQ Problem Determination presentation: 01.ibm.com/support/docview.wss?uid=swg27009878

**First-failure Support Technology (FST), First-failure Data Capture (FDC)**

- Intended to log enough information about unexpected events (not routine MQ errors) that the problem can be resolved without further recreation and tracing.
- Located in the top-level errors directory, plain text format, never purged by MQ.
- Named like AMQnnnnn.x.FDC
- Probe severity: 1 = Critical, 2 = Error, 3 = Warning, 4 = Informational
- Issue the ffstsummary command from the errors directory to get a summary listing
- IBM Hursley lab article on FFST files: https://hursleyonwmq.wordpress.com/2007/05/04/introduction-to-ffsts/
Tracing

- MQ tracing can be started and stopped from the command line, and also from MQ Explorer.
- Command-line options allow you to choose the desired level of detail
- Output goes to the "trace" subdirectory at the top of the MQ tree
- One active trace output file per MQ process; suffixes .TRC and .TRS are used for rollover (.TRC is more recent)
- Unix requires an additional step, to format the trace output into human-readable form (.FMT files)
- New in MQ v7: strmqtrc -c to start tracing, and automatically stop after an FDC when a specific Probe ID is generated
- Tracing and debugging 2035 authorization failures:

Commands to Enable and Disable Tracing

- Enable tracing: strmqtrc
- Reproduce the problem
- End tracing: endmqtrc
- On Unix: use dspmqtrc to translate binary trace output files to text format
- Result: text files with names ending in .TRS and .TRC on Windows; binary .TRS and TRC and human-readable .FMT files on Unix

Real Time Monitoring

- Checking queue manager and channel statistics while MQ is running
- Must be enabled before MQ will start recording data (default is not to collect most of this information)
- Queue manager attributes MONQ, MONCHL
  - NONE = disabled, no matter what the queues/channels say
  - OFF= off, but individual queues and channels can override
  - LOW, MEDIUM, HIGH = enabled, individual queues and channels can override
- Queue attribute MONQ and channel attribute MONCHL
• QMGR = use the queue manager attribute setting
  • OFF, LOW, MEDIUM, HIGH (LOW, MEDIUM, and HIGH are equivalent for queues
  • Defaults are queue manager OFF, queue and channel = QMGR
• runmqsc
  • DISPLAY QSTATUS (queueName)
  • DISPLAY CHSTATUS (channelName)
• MQ Explorer: right-click the queue name, click Status
• Fields
  • MSGAGE: age of oldest message on the queue, in seconds
  • QTIME: average time in microseconds between put and get (recent average and long-term average)
  • LGETTIME and LGETDATE: time/date of last get operation
  • LPUTTIME and LPUTDATE: time/date of last put operation
  • UNCOM: pending uncommitted puts and gets
• Some queue status attributes do not require monitoring to be enabled:
  • CURDEPTH: current queue depth (number of messages on the queue)
  • IPPROCS, OPPROCS: number of processes that have the queue open for input (can get messages) and for output (can put messages)
  • DISPLAY QL (queueName) CURDEPTH IPPROCS OPPROCS
• MONCHL=off
  • STATUS; MCASTAT, SUBSTATE: channel and MCA state information
  • CURSEQNO: sequence number of last message sent or received
  • BTYSSENT, BYTSRCVD: number of bytes sent and received since the channel was started
  • MSGS: number of messages sent or received since the channel was started
  • LSTMSGTI, LSTMSGDA: time and date of last message sent or received
• MONCHL=enabled
  • NETTIME: recent and long-term average network round-trip times in microseconds for request/response to/from the other end of the channel
    • Requires MONCHL = MEDIUM or HIGH
  • XQTIME: average times in microseconds that messages were on the transmission queue before being retrieved
    • Requires MONCHL = HIGH
    • Sender channels only (same with NETTIME)
Event Monitoring

- An instrumentation event is a logical combination of events that is detected by a queue manager or channel instance. Such an event causes the queue manager or channel instance to put a special message, called an event message, on an event queue.

- Event messages go to one of a small set of system-defined event queues (SYSTEM.ADMIN.*.EVENT), depending on their type. Event message payloads are in binary format, not human-readable text.

- **Decode**
  - There is a sample program in the InfoCenter to partially decode them, and you could build on that program; OR
  - Use Support Pac MS0P: an extension to MQ Explorer that decodes event messages into readable text
  - Windows Perfmon can also be used to visually monitor queue depth

- **Queue Depth**
  - Queue depth events, a type of performance event, will show up in the SYSTEM.ADMIN.PERFM.EVENT queue
  - Documented here:
  - Enable PERFMEV on the queue manager
  - Enable some or all of QDPMAXEV, QDPHIEV, QDPLOEV on the queue
  - Set MAXDEPTH, QDEPTHHI, QDEPTHLO (the last two are percentages) on the queue

```
ALTER QMGR PERFMEV (ENABLED)
DEFINE QLOCAL (MY_Q)
ALTER QL (MY_Q) MAXDEPTH (10) QDPMAXEV (ENABLED) + QDEPTHHI (50) + QDPHIEV (ENABLED) + QDEPTHLO(30) QDPLOEV (DISABLED)
```

- Now put messages on the queue (I attempted to put 11 messages, using amqsput; the 11th put failed, of course)
- CURDEPTH of SYSTEM.ADMIN.PERFM.EVENT is incremented after the 5th and the 11th put

**MS0P**

- Installation is just a matter of unzipping into the right place, modifying one text file, then strmqcfg -c
- After that, you can right-click a queue manager, then do Event Messages > Format Events...
• Can watch individual queues, showing number of puts and gets, plus bargraph of queue depth, every N seconds (configurable via Window > Preferences)

**Not Authorized Events**

• "Queue manager events" include six types of "not-authorized" events
• Messages appear in SYSTEM.ADMIN.QMGR.EVENT
• To enable: ALTER QMGR AUTHOREV (ENABLED)

**Put and Get Programs**

• "Bindings mode" (communicate with queue manager via IPC, works only on the queue manager machine): amqsput, amqsget
• "Client mode" (uses TCP and MQ channels, works from remote machines too): amqsputc, amqsgetc
• Command-line arguments: queue name and queue manager name; e.g. amqsput ORDERQ QM_1
• Need full path (e.g. /opt/mqm/samp/bin/amqsput) on Unix
• "Put" programs allow you to type text, sending one message for each line; "get" programs retrieve and display messages

**SupportPacs**

• A few useful SupportPacs:
  • IH03 (RFHutil): GUI to put and get messages, decode and display message headers, etc
  • MO04: SSL setup wizard
  • MQ Health Checker

**Message Monitoring**

• The process of identifying the route a message has taken through a queue manager network
• Can be done in two ways:
  • Setting a flag in any MQ message can cause special "activity report" messages to be generated; or
  • Special "trace-route" messages can be sent; activity information is accumulated in the message payload
• The dspmqrte program uses these techniques to trace message flow through an MQ network
• SupportPac MS0P also has trace-route functionality
• Setup SOURCE and TARGET queue managers
• Right-click Q.ON.TARGET (a remote queue definition on queue manager SOURCE) in MQ Explorer, select Trace Route
• Reference: http://www.ibm.com/support/knowledgecenter/SSFKSJ_7.5.0/com.ibm.mq.mon.doc/q036600_.htm

Retry on Server Down
• To retry for server going down (e.g. reason code 2162): Application Servers > $SERVER > Message Listener Service > Content > Additional Properties > Custom Properties
  • MAX.RECOVERY.RETRIES=N
  • RECOVERY.RETRY.INTERVAL=60

Troubleshooting WXS

Hung Thread Detection
WXS has hung thread detection similar to that available in WAS. For example:

Stack Trace:
com.ibm.ws.classloader.CompoundClassLoader.loadClass(CompoundClassLoader.java:549)
  java.lang.ClassLoader.loadClass(ClassLoader.java:357)
  com.ibm.ws.xs.util.XSUtilities.loadClass(XSUtilities.java:77)
com.ibm.ws.xs.io.ObjectStreamPool$ClassForNamePrivilegedAction.run(ObjectStreamPool.java:467)
  java.io.ObjectInputStream.readNonProxyDesc(ObjectInputStream.java:1610)
  java.io.ObjectInputStream.readClassDesc(ObjectInputStream.java:1515)
  java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1769)
  java.io.ObjectInputStream.readObject0(ObjectInputStream.java:1348)
  java.io.ObjectInputStream.readObject(ObjectInputStream.java:370)
  java.util.HashMap.readObject(HashMap.java:1155)
sun.reflect.GeneratedMethodAccessor22.invoke(Unknown Source)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
  java.lang.reflect.Method.invoke(Method.java:606)
java.io.ObjectStreamClass.invoke ReadObject(ObjectStreamClass.java:1017)
java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1891)
java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1796)
java.io.ObjectInputStream.readObject(ObjectInputStream.java:1348)
java.io.ObjectInputStream.readSerialData(ObjectInputStream.java:1913)
java.io.ObjectInputStream.readOrdinaryObject(ObjectInputStream.java:1796)
java.io.ObjectInputStream.readObject(ObjectInputStream.java:1348)
java.io.ObjectInputStream.readObject(ObjectInputStream.java:370)
com.ibm.ws.objectgrid.datagrid.BaseAgentCommand.inflateAgent(BaseAgentCommand.java:323)
com.ibm.ws.objectgrid.datagrid.BaseAgentCommand.setBaseMap(BaseAgentCommand.java:173)
com.ibm.ws.objectgrid.server.impl.ServerCoreEventProcessor.processCommand(ServerCoreEventProcessor.java:1454)
com.ibm.ws.objectgrid.server.impl.ServerCoreEventProcessor.processClientServerRequest(ServerCoreEventProcessor.java:2596)
com.ibm.ws.objectgrid.server.impl.ShardActor.receive(ShardActor.java:333)
com.ibm.ws.xs.xio.actor.impl.XIOReferableImpl.dispatch(XIOReferableImpl.java:110)
com.ibm.ws.xs.spi.xio.actor.XIORegistry.sendToTarget(XIORegistry.java:977)
com.ibm.ws.xs.xio.transport.channel.XIORegistryRunnable.run(XIORegistryRunnable.java:88)
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
com.ibm.ws.objectgrid.thread.XSThreadPool$Worker.run(XSThreadPool.java:309)
Runnable: com.ibm.ws.objectgrid.util.security.SecurityContextRunnable@5fa09130.
Unable to Load Data into the Grid

**Strategy 1: Object grid failed to start**

Execute the command `$WXS_HOME/bin/xsadmin.sh -primaries`

Two common culprits are:

1. Firewall rules blocking something that shouldn't be
2. The numInitialContainers value is set higher than the total number of containers being started (in deploy.xml).

**Appendix**

**Resources**

- WebSphere Application Server Performance Team: [http://www-01.ibm.com/software/webservers/appserv/was/performance.html](http://www-01.ibm.com/software/webservers/appserv/was/performance.html)

**Opinions**

**IBM HeapAnalyzer versus the IBM Memory Analyzer Tool**

*by Kevin Grigorenko*

IBM HeapAnalyzer (HA) is a popular heap analysis tool. HeapAnalyzer pioneered some of the original leak suspect heuristics and it has a very easy to use interface. However, while HeapAnalyzer supports HPROF dumps and IBM system dumps, it was primarily designed for IBM PHDs. This contributes to its simplicity but also constrains its feature set. Some consider this a good thing as most dump analysis just needs a straightforward analysis of leak suspects, so time-to-load and ease-of-use are key features.
However, the IBM Memory Analyzer Tool (MAT) with system dumps can solve the same set of problems better (not to mention it can solve a whole different set of problems that HeapAnalyzer can't easily). For example, HeapAnalyzer may tell you that class X is consuming 50% of the heap and caused the OOM, but MAT may tell you the same thing and that it occurred on a particular thread in a particular stack frame from a particular application. Having memory contents and stack frame locals opens up whole new worlds of diagnosis. Finally, MAT is an officially supported tool by the IBM Java team. If you are new to heapdump analysis, I recommend MAT. If you are used to HeapAnalyzer, I recommend you continue to use it for simple problems, but use MAT for problems HA doesn't solve, or when the owner of the culprit code pushes back on the HA analysis or doesn't know what to do with it. Ultimately, I recommend you move towards MAT (it has similar leak suspect heuristics) because using this tool frequently (especially with system dumps) may expand your mind when tackling future problems.

**IBM Installation Manager (IM)**


**Offline Installations**


**imcl**


**Help**

Invoke help: $ imcl help

For example to list the parameters and options being accepted by updateAll: $ imcl help updateAll

**List Installed Packages**

$ imcl listInstalledPackages
com.ibm.websphere.ND.v80_8.0.9.20140530_2152
com.ibm.websphere.PLG.v80_8.0.3.20120320_0536
com.ibm.websphere.IHS.v80_8.0.3.20120320_0536

**List Installed Features of a Package**

$ imcl listInstalledFeatures com.ibm.websphere.ND.v80_8.0.9.20140530_2152
Other Examples

List available packages and features in a repository: $ imcl listAvailablePackages -repositories /repository.config -features -long

Installing WAS8 with 64Bit Java: $ imcl install com.ibm.websphere.ND.v80,core.feature,ejbdeploy,com.ibm.sdk.6_64bit -repositories /disk1 -installationDirectory -accessRights nonAdmin -acceptLicense -log /tmp/WAS8_install.log [-sharedResourcesDirectory ]

Installing an iFix (PM48831): $ imcl install 8.0.0.0-WS-WASND-IFPM48831_8.0.0.20111110_1512 -installationDirectory -acceptLicense -log /tmp/WAS8_iFix_install.log -repositories [-sharedResourcesDirectory ]

Note: You might need to run imcl listAvailablePackages to determine the [-sharedResourcesDirectory ] of the iFix

Uninstalling an iFix (PM48831): $ imcl uninstall 8.0.0.0-WS-WASND-IFPM48831_8.0.0.20111110_1512 -installationDirectory -log /tmp/IFPM48831_uninstall.log

Running a fill installation of course works as well - however the properties list is depending on the product being used. As most products provide a sample response file it's the easiest way to determine the properties from there. Look at the "" lines in the respone files. Each "key-name" is converted to a property. Note: As properties are seperted by "," you have to double the "," in the "key-name" if the "key-name" contains commas.

Save Credentials

If a repository requires authentication, you must save the password to a local file: http://www-01.ibm.com/support/knowledgecenter/SSDV2W_1.8.1/com.ibm.cic.commandline.doc/topics/t_imcl_store_credentials.html

First, create a master password file with a plain text password:

```
$ cat > master_password_file.txt
${PASSWORD}
^D
```

Next, create the credential file:

```
$ imutilsc saveCredential -url ${URL} -userName ${USER} -userPassword ${PASSWORD} -secureStorageFile credential.store -masterPasswordFile master_password_file.txt
Successfully saved the credential to the secure storage file.
```

Use the `-secureStorageFile` and `-masterPasswordFile` imcl options to specify these files. For example:
$ imcl -acceptLicense -secureStorageFile credential.store -masterPasswordFile master_password_file.txt updateAll

If these passwords and files are sensitive, remove them after the operations are complete and clear your shell history.

Update Package

One way to update a package is to enable only the repositories with those package updates (this can be done easily under Preferences in console mode) and then use the `updateAll` command. For example:

$ imcl -acceptLicense -secureStorageFile credential.store -masterPasswordFile master_password_file.txt updateAll

Updated to com.ibm.websphere.IHS.v80.8.0.10.20150116_1534 in the
/opt/IBM/HTTPServer directory.
Updated to com.ibm.websphere.ND.v80.8.0.10.20150116_1534 in the
/opt/IBM/WebSphere/AppServer directory.
Updated to com.ibm.websphere.PLG.v80.8.0.10.20150116_1534 in the
/opt/IBM/WebServer/Plugins directory.

Console Mode

Console mode is a feature of `imcl` which lets you navigate through IM like you would through the GUI but through a shell:

$ imcl -c

=====> IBM Installation Manager

Select:
1. Install - Install software packages
2. Update - Find and install updates and fixes to installed software packages
3. Modify - Change installed software packages
4. Roll Back - Revert to an earlier version of installed software packages
5. Uninstall - Remove installed software packages

Other Options:
L. View Logs
S. View Installation History
V. View Installed Packages
P. Preferences
A. About IBM Installation Manager
X. Exit Installation Manager

Installing Fix Packs and i-Fixes

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.installation.nd.doc/ae/tins_i
POSIX

Shells

There is a convention that if the logged in user is root, then the shell prompt character is #, while non-root users show $.

To print which shell you're currently using, try these commands:

```bash
$ echo $0
bash
$ echo $SHELL
/bin/bash
$ ps -p $$
   PID TTY          TIME CMD
     6549 pts/4    00:00:00 bash
```

bash

Change the command prompt to include more information:

```bash
$ export PS1="[\u@\t \w]\$ 
```

Search command history using Ctrl+R and typing and recall with ESC or execute with ENTER:

```bash
$ less /etc/hosts
$ othercommand1
$ othercommand2
$ # Now type Ctrl+R and type less
(reverse-i-search)`less': less /etc/hosts
```

Recall and execute the last command with !!:

```bash
$ /opt/IHS/bin/apachectl restart
$ sudo !!
```

Recall and execute the first previous command containing a search string with !?:

```bash
$ /opt/IHS/bin/apachectl restart
$ !!apachectl
```

Recall and execute the last command but replace all instances of one expression with another:

```bash
$ !!:gs/stop/start/
```

Use the last argument of the previous command with !!:$, or all arguments of the previous command with !!:*:

```bash
$ !?:apachectl
```

Search history and re-run a particular command based on its position in history:

```bash
$ history | grep startServer.sh
527 /opt/IBM/WebSphere/profiles/node01/bin/startServer.sh server1
```
Print the name of the current user:

```
$ whoami
```

Extract all .tar.gz files into subdirectories using the name of the file without the extension:

```
$ for i in *.tar.gz; do mkdir `basename $i .tar.gz` && mv $i `basename $i .tar.gz` && pushd `basename $i .tar.gz` && tar xzvf $i && rm $i && popd; done
```

Extract all .tar files into subdirectories using the name of the file without the extension:

```
$ for i in *.tar; do mkdir `basename $i .tar` && mv $i `basename $i .tar` && pushd `basename $i .tar` && tar xvf $i && rm $i && popd; done
```

Extract all .zip files into subdirectories using the name of the file without the extension:

```
$ for i in *.zip; do mkdir `basename $i .zip` && mv $i `basename $i .zip` && pushd `basename $i .zip` && unzip $i && rm $i && popd; done
```

Gunzip all .gz files in a directory:

```
$ find . -type f -name "*gz" -print | while read line; do pushd `dirname $line`; gunzip `basename $line`; popd; done
```

Change to a directory by replacing a part of the current directory:

```
cd ${PWD/Dmgr01/AppSrv01}
```

Recall the last word of the previous command using Alt+. 

### Global aliases

For truly global aliases, update the scripts for both interactive shells (/etc/profile.d/*) and non-interactive shells (/etc/bashrc or /etc/bash.bashrc, depending on the distribution).

First, create a shell script with the commands you want to run and place it in a common location such as /etc/globalprofile.sh:

```bash
#!/bin/sh
alias x="exit"
alias l="ls -ltrh"
export PS1="[\u@	 \w]$ "
```

Then add the execute permission:

```
# chmod +x /etc/globalprofile.sh
```

Finally, append the following line to both interactive and non-interactive shell script locations (e.g. /etc/bashrc):

```
source /etc/globalprofile.sh
```
Truncating Logs
While some operating systems have commands specifically for truncation (e.g. "truncate" on Linux), it is simpler and more cross-platform to simply write /dev/null on top of a file to truncate it:

```bash
cat /dev/null > file.log
```

This does not work with sudo because the redirection operator occurs outside of the sudo. In that case, you can use tee:

```bash
cat /dev/null | sudo tee file.log
```

Defunct Processes
Use "ps -elf | grep defunct" to monitor defunct processes.

"Processes marked <defunct> are dead processes (so-called "zombies") that remain because their parent has not destroyed them properly." ([http://man7.org/linux/man-pages/man1/ps.1.html](http://man7.org/linux/man-pages/man1/ps.1.html))

"A defunct process, also known as a zombie, is simply a process that is no longer running, but remains in the process table to allow the parent to collect its exit status information before removing it from the process table. Because a zombie is no longer running, it does not use any system resources such as CPU or disk, and it only uses a small amount of memory for storing the exit status and other process related information in the slot where it resides in the process table." ([http://www-01.ibm.com/support/docview.wss?uid=isg3T1010692](http://www-01.ibm.com/support/docview.wss?uid=isg3T1010692))

Defunct processes are processes that have exited and are waiting for the parent process to read its exit code. Most of the resources of the exited process are released; however, the PID, exit code, and process table entries are still resident and a persistent and large number of defunct processes can limit scalability. Every process will be defunct, but normally it is only for a short period of time. Normally, persistent defunct processes mean that the parent process is hung. In the case of WAS, this is usually the nodeagent process. To remove defunct processes, kill the parent process. Before doing this, gather diagnostics on the parent process such as performing activity on it to see if it is still alive, requesting a thread dump, and finally requesting a core dump. Killing the parent process will cause the parent process of the defunct process to become the init (1) process which will then read the exit code and allow the defunct process to finish.

SSH
To bypass any configured private keys:

```bash
$ ssh -o PubkeyAuthentication=no user@host
```

SSH Port Forwarding
Often you can SSH into a box but other ports that you want to access are blocked by firewalls. You can create an SSH tunnel that takes traffic on your machine at some port, forwards it through an SSH connection and sends it to a different port on the target server. For example, let's say hostX has something that is listening on port 8879 which you can't access from outside that box. You can create a tunnel like this:
$ ssh -L 9999:hostX:8879 sshuser@hostX

Now you should have a listening port on localhost port 9999. You can access this port through your client program as you would access port 8879 on hostX.

This can also be done with programs such as putty by using the tunnel option:

![PuTTY Configuration](image)

**kill**

kill is used to send signals to processes. The general format of the command is:

```bash
$ kill -${SIGNAL} ${PID}
```

${SIGNAL} is either a number or the name of the signal.

For example, to send the equivalent of Ctrl+C to a process 123:

```bash
$ kill -INT 123
```

**less**

less is a common command to browse files and input:

```bash
$ less input
```
Tips:

- Jump to the beginning of input: g
- Jump to the end of input: G
- Jump to a line number N: Ng
- Go to the next input: :n
- Search for something: /search
- Find next: n
- If you jump to the end of input and it says "Calculating line numbers," press Ctrl+C if you don't need to do this to stop the calculation.
- Start tailing a log: Shift+F

To show line numbers, use -N:

```bash
$ less -N input
```

**tail**

tail may be used to skip the first N lines of input using `-n (N+1)`. For example, to skip the first line of input:

```bash
$ tail -n +2 input
```

**sort**

Sort by a particular column using `-k`:

```bash
$ sort -k 3 input
```

Sort numerically:

```bash
$ sort -k 3 -n input
```

**awk**

awk is often used as a simple way to do line-by-line manipulation. For example, to print the 4th column in all piped in lines:

```bash
$ cat input | awk '{print $4}'
```

Additionally, the `$0` variable represents the entire line:

```bash
$ cat input | awk '{print $4,$0}'
```

**bc**

paste and bc may be used to sum a set of numbers from input:

```bash
$ cat input | paste -sd+ | bc
```
**sed**

sed and bc may be used to do simple math on input:

$ cat input | sed 's/$*/1024/' | bc

**Perl**

perl is a commonly used scripting language. A perl script normally has the .pl extension and starts with this shebang line:

```
#!/usr/bin/env perl
```

Useful command line options:

- `perldoc perlrun`: Man page of executing the perl interpreter.
- `-e`: Specify perl code on the command line rather than a perl file.
- `-p`: Run specified perl command on each line of standard input.
- `-n`: Same as `-p` except that each line is not also printed.

For example, to convert a POSIX date epoch into a human-readable time:

$ date +%s | perl -ne 's/\d+/localtime($1)/e;'

The `$_` variable will contain each line of the file.

Code may be run at the start and end using `BEGIN {}` and `END {}` blocks, respectively:

$ date +%s | perl -ne 'BEGIN { print "Starting\n"; } s/\d+/localtime($1)/e; END { print "Finished\n"; }'

Useful things to remember in perl:

- `$x =~ /$REGEX/`: Return true if $REGEX matches $x
- `$x =~ s/$REGEX//g`: Replace all occurrences of $REGEX in $x with nothing.

Commonly used regular expression tokens:

- Match zero or more: *
- Match one or more: +
- Any character: .
- White space character (space, tab, or newline): \s
- Opposite of white space character: \S
- Word character (a-z, A-Z, 0-9, or underscore): \w
- Non-word character: \W
- Digit character (0-9): \d
- Non-digit character: \D

**wget**

wget may be used to execute an HTTP request:

$ wget http://ibm.com/
Saving to: “index.html”
When multiple URLs are passed to wget, if possible, wget will attempt to re-use the same TCP socket. Use Perl to automate generating the same URL many times on the command line. In the following example, 64 requests will be attempted over the same socket:

```bash
$ wget -0/dev/null `perl -e 'print "http://ibm.com/ " x 64;\n'
```

To review response headers and (short) bodies, a useful one-liner is:

```bash
$ wget -qS http://ibm.com/ -O-
```

**netcat (nc) / openssl s_client**

When you want more control over what goes into the HTTP/HTTPS request, you can use printf and netcat or openssl s_client:

```bash
$ printf "GET / HTTP/1.1\r\nHost: example.com\r\n\r\n" | nc 0 80
$ printf "GET / HTTP/1.1\r\nHost: example.com\r\n\r\n" | openssl s_client -connect 0:443 -ign_eof
```

**find**

The /usr/bin/find command searches for files recursively based on their name or metadata. Check the bottom of the Linux manual for examples.

```bash
$ find /opt/IBM/WebSphere -name server.xml
$ find /opt/IBM/WebSphere -size +100M (note: the M suffix is not portable)
$ find . -name server.xml|grep -vi Templates|xargs grep startupTraceSpecification
```

**gpg**

**File Encryption**

Encrypt a file for storage or transit:

```bash
$ gpg --s2k-mode 3 --s2k-count 65536 --force-mdc --cipher-algo AES256 --s2k-digest-algo sha512 -o ${OUTPUTFILE}.pgp --symmetric ${INPUTFILE}
```

**File Decryption**

Decrypt a PGP-encrypted file:

```bash
$ gpg --output ${OUTPUTFILE} --decrypt ${INPUTFILE}.pgp
```

**touch**

changes the timestamp of a file (=access/modification time) to the current date and time:

```bash
$ touch input
```
Tip: "touch [non-existing filename]" will create a new empty file (no directories). You can also create several new files using "touch [newfile1 newfile2 newfile3]"

Filenames and special characters
Special characters in the terminal include $ < > & ;"\nIf you'd like to use them, you'll need to precede "escape" them with a \ [back slash]. There is no way you can create a filename with a / [forward slash] or null character.

Auto completion
Use "cd + [first few letters of your filename] + TAB {+ TAB}" to change directory to files starting with the letters you specified using auto completion:

   $ cd + inp + TAB {or TAB + TAB to display all options}

"ls + TAB + TAB" will open up a list of suggestions which you can use with a given command.

Keyboard shortcuts:
Tips:

• Move to the beginning of a line: CTRL+A
• Move to the end of a line: CTRL+E
• Move one word backwards at a time: ALT+B
• Delete character at cursor location: CTRL+D
• Cut text from the cursor location to the end of the line (beginning of the line): CTRL+K (U) and use CTRL+Y to paste it back; "kill" text and "yank" it.
• Make the current word [at the beginning of a word] lowercase (uppercase): ALT+L (U)

   tac
Show contents of filename1 and filename2 in a reverse order:

   $ tac [filename1][filename2]

Types of commands
To list all available built-in shell commands for a particular shell use:

   $ compgen -b

To display your command type you can use "type [command name] OR file [absolute path to command]":

   $ type cd
   OR type ifconfig and file /sbin/ifconfig

Alternatively, you can use the "which" command to display an executable location/the absolute command path (not working for shell built-ins):
$ which ifconfig

To check if a command name is already in use or may be used as an alias you may use "type [potential alias name]":

$ type myalias

Make your own commands

• To combine 1stcommand and 2ndcommand etc., use ”1stcommand; 2ndcommand;....” Correct commands will be executed unless you use exit:

$ date; cal

• Alternatively, you can use "&amp;&amp;" to combine commands. However, this will only execute until it encounters an error ("short circuit evaluation"): 

$ date &amp;&amp; cal

• Wildcards:
  • Represents or matches any characters: *
  • Represents or matches a single character: ?
  • Match a range of characters: [range of characters]
  • Not match a range of characters: ![range of characters]
  • Any number from numbers of the digit range: [digit-digit]*
  • Uppercase: [:upper:]
  • Lowercase: [:lower:]
  • Digit: [:digit:]
  • Alphabetical: [:alpha:]
  • Alphanumeric: [:alnum:]
  • NB: To negate this use ![....:]

Internet Domains

Reserved Domains

example.com is a commonly used test domain: https://tools.ietf.org/html/rfc2606

HTML

HTML5 specification: http://www.w3.org/TR/html5/single-page.html

Example HTML5 Page

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Title</title>
  </head>
</html>
```
OpenLDAP

OpenLDAP Software is an open source implementation of the Lightweight Directory Access Protocol: http://www.openldap.org/

The older slapd.conf file is deprecated, and the newer configuration files under slapd.d should be edited using LDAP utilities rather than manually: http://www.openldap.org/doc/admin/slapdconf2.html

Configuration

Dump all configuration:

```bash
$ ldapsearch -Y EXTERNAL -H ldapi:/// -b "cn=config"
```

Configuration is modified by creating an LDAP Data Interchange Format (LDIF) file with the desired changes and running:

```bash
$ ldapmodify -Y EXTERNAL -H ldapi:/// -f ${file}.ldif
```

In recent versions, the main configuration is contained in LDIF files under some directory such as /etc/openldap/slapd.d/cn=config; however, these files should not be edited directly. Instead, create an LDIF file with the changes and run ldapmodify.

For example, in a simple configuration, it is common to change olcSuffix, olcRootDN, olcRootPW, and olcAccess. Create an update_configuration.ldif file, replace dc=example,dc=com with your domain, and run `slappasswd` to generate the input for olcRootPW:

```bash
dn: olcDatabase={0}config,cn=config
changeType: modify
replace: olcRootPW
olcRootPW: {SSHA}ugwz71gwNPJuW5bQzyqIMATp8wOpu7Io
```

```bash
dn: olcDatabase={2}bdb,cn=config
changeType: modify
replace: olcSuffix
olcSuffix: dc=example,dc=com
```

replace: olcRootDN
olcRootDN: cn=Manager,dc=example,dc=com
- replace: olcRootPW
  olcRootPW: {SSHA}ugwz71gwNPJuw5bQzyqIMATp8wOPu7Io
-
  dn: olcDatabase={1}monitor,cn=config
  changetype: modify
  replace: olcAccess
   olcAccess: {0}to * by
   dn.base="gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth"
     read by dn.base="cn=Manager,dc=example,dc=com" read by * none
-

Input this file to ldapmodify:

$ ldapmodify -Y EXTERNAL -H ldapi:/// -f update_configuration.ldap

LDAP Data Interchange Format (LDIF)

LDIF is specified through RFC 2849: https://tools.ietf.org/html/rfc2849

The general form is:

# Comment
key: value
  continuation

- A continuation occurs when a line starts with one space. That one space is removed and the rest
  is concatenated to the previous line. Therefore, it's almost always necessary to use two spaces so
  that there is a space between the concatenation.
- If a key is followed by two colons, the value is Base-64 encoded.
- When using `ldapmodify`, operations are separated by a line with a dash in it, followed by a
  blank line. This does not apply to `ldapadd`.

ldapadd

Instead of creating an LDIF file beforehand, you may omit `-f`, enter the LDIF in the standard input
and then type Ctrl+D. For example:

$ ldapadd -D cn=Manager,dc=example,dc=com -w password
dn: ...
Ctrl+D

Example: Create Organization

Here is an example create_organization.ldif with a single user:

dn: dc=example,dc=com
objectClass: dcObject
objectClass: organization
dc: example
Then add all of the items with:

```
$ ldapadd -f create_organization.ldif -D cn=Manager,dc=example,dc=com -W
```

**Example: Add User**

```
$ ldapadd -D cn=Manager,dc=example,dc=com -w password
dn: cn=Admin,ou=Users,dc=example,dc=com
  cn: Admin
  sn: Admin
  objectClass: inetOrgPerson
  userPassword: {SSHA}baYn/l/wd41jpw5k0GvSPn99DboceyQZ
  uid: 2
```

**ldapsearch**

Example output:

```
$ ldapsearch -LLL -x -b 'dc=example,dc=com' '(objectclass=*)'
dn: dc=example,dc=com
  objectClass: dcObject
  objectClass: organization
dc: example
  o: example

  dn: ou=Users,dc=example,dc=com
  objectClass: organizationalUnit
  ou: Users

  dn: cn=User1,ou=Users,dc=example,dc=com
  cn: User1 LastName
```
Example: Find Users by Attribute

$ ldapsearch -LLL -x -b 'dc=example,dc=com' '(&(sn=LastName) (objectclass/inetOrgPerson))'

dn: cn=User1,ou=Users,dc=example,dc=com
  cn: User1 LastName
  cn: User1
  sn: LastName
  objectClass: inetOrgPerson
  uid: 1
  userPassword:: e1NTSEF9M0FjcXdzMFVPRmlSQ1Z2cGZaR3JQUWczNXRsejhOMng=

Example: Find Groups that Contain a User

$ ldapsearch -LLL -x -b "dc=example,dc=com" -D cn=Manager,dc=example,dc=com -w password "(&(objectclass=groupOfNames) (member=cn=User1,ou=Users,dc=example,dc=com))"

dn: cn=Group1,ou=Users,dc=example,dc=com
  cn: Group1
  objectClass: groupOfNames
  member: cn=User1,ou=Users,dc=example,dc=com

ldapmodify

Instead of creating an LDIF file beforehand, you may omit `-f`, enter the LDIF in the standard input and then type Ctrl+D. For example:

$ ldapmodify -D cn=Manager,dc=example,dc=com -w password
dn: ...
  changetype: ...
  Ctrl+D

Example: Modify User Password

$ ldapmodify -D cn=Manager,dc=example,dc=com -w password
dn: cn=User1,ou=Users,dc=example,dc=com
  changetype: modify
replace: userPassword
userPassword: {SSHA}3Acqws0UFiRCVvpfZGrPQg35tlz8N2x
^D

Example: Add Member to Existing Group
$ ldapmodify -D cn=Manager,dc=example,dc=com -w password
dn: cn=Group1,ou=Users,dc=example,dc=com
changetype: modify
add: member
member: cn=Admin,ou=Users,dc=example,dc=com
^D

ldapwhoami
Use `ldapwhoami` to test user credentials.
Example success:
$ ldapwhoami -vvv -D "cn=User1,ou=Users,dc=example,dc=com" -x -w password
ldap_initialize( <DEFAULT> )
 dn:cn=User1,ou=Users,dc=example,dc=com
Result: Success (0)
Example failure:
$ ldapwhoami -vvv -D "cn=User1,ou=Users,dc=example,dc=com" -x -w test
ldap_initialize( <DEFAULT> )
ldap_bind: Invalid credentials (49)

Wily Introscope
- introscope.agent.disableAggregateCPUUtilization=true
- introscope.agent.sqlagent.sql.maxlength=[1-990]
- introscope.autoprobe.dynamicinstrument.enabled=false
- introscope.agent.remotedynamicinstrumentation.enabled=false
- introscope.autoprobe.logfile=logs/AutoProbe.log
  - log4j.appenders.logfile=/dev/null
  - log4j.logger.IntroscopeAgent=OFF

- transport.outgoingMessageQueueSize=6000
- transport.override.isengard.high.concurrency.pool.min.size=10
- transport.override.isengard.high.concurrency.pool.max.size=10
OpenOffice/LibreOffice

Tips
1. If cells contain numbers that start with an apostrophe (‘), then things such as aggregate functions in pivot tables will not work, even if the cells are formatted as "Number." To remove the apostrophes: Edit > Find & Replace > Search for = ".*", Replace = "&", More Options > Check "Regular Expressions"

Acronyms
• RAS: Reliability, Availability, Serviceability

Revision History
This version was built on 2019-02-25T10:03:42.520-0800.

1.0.29 (February 2019)
• Add Containers chapter

1.0.28 (January 2019)
• Add AIX native memory debug info

1.0.27 (November 2018)
• Add Linux eBPF example for native memory leaks

1.0.26 (November 2018)
• Add Linux lsof

1.0.25 (November 2018)
• Add Linux perf On-CPU stack sampling with wallclock timestamps

1.0.24 (October 2018)
• Add Traditional WAS page to discuss Startup

1.0.23 (October 2018)
• Update GCMV installation instructions
1.0.22 (October 2018)
- Added information on tuning DirectByteBuffer pool sizes for both editions of WAS
- Describe how to request exclusive access for a system dump on IBM Java to avoid dumps taken during GC

1.0.21 (May 2018)
- Added Mac section
- Added Linux Available memory reference

1.0.20 (March 2018)
- Added pureScale section

1.0.19 (January 2018)
- Add WAS Traditional instructions on setting up ODR custom logging

1.0.18 (September 2017)
- Update MAT instructions with DTFJ Java 8 note.

1.0.17 (August 2017)
- Update Memory Analyzer download instructions.

1.0.16 (April 2017)
- Change IBM Java -Xverbosegclog recommendation

1.0.15 (April 2017)
- Add Linux swappiness, OOM killer, and swap recommendations

1.0.14 (January 2017)
- Change major tool instructions to use Eclipse instead of ISA

1.0.13 (June 2016)
- Fix broken Knowledge Center links.
- Change WAS Classic to WAS Traditional

1.0.12 (February 2016)
- Various updates based on field work.
1.0.11 (December 2015)
• Fix error referencing META-INF/lib instead of META-INF/resources in WAS > WAS Classic > HTTP, in the section "ServletContext.getResource performance"

1.0.10 (December 2015)
• Change graphs from R to gnuplot

1.0.9 (December 2015)
• Update Solaris KSSL guidance.

1.0.8 (December 2015)
• Add more TPV/PMI screenshots.
• Add more Health Center screenshots.
• Add Liberty request timing and event logging details.

1.0.7 (December 2015)
• Update GCMV page.

1.0.6 (September 2015)
• Add troubleshooting recipes.

1.0.5 (August 2015)
• Rename WAS Traditional Profile to WAS Classic

1.0.4 (May 2015)
• Remove unused Liberty idle tuning option.

1.0.3 (April 2015)
• Rename Java chapters.

1.0.2 (April 2015)
• Add example Solaris DTrace scripts and `vmstat -p` information.

1.0.1 (February 2015)
• Add February SPECj benchmark results.
1.0.0 (January 2015)

- First public version.

Notices

Statement of Support

While IBM welcomes any comments or suggestions, this Cookbook is not supported by IBM and is provided on an "as-is" basis without warranty of any kind. IBM may make updates if needed and as time permits.

The Cookbook contains techniques and tools that may not be supported by IBM's support process. For example, you may gather some form of trace that captures a problem that the support process is not accustomed to analyzing. If you have any question about any content or recommendations, particularly if you are about to do something in a production environment, please first open a PMR to ask what is and isn't supported.

Terms of Use


References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, New York  10594 USA

Trademarks and Service Marks

IBM, the IBM logo and ibm.com are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at http://www.ibm.com/legal/us/en/copytrade.shtml