IBM WebSphere Application Server Network Deployment for Distributed Platforms, Version 8.5

Tuning guide

Note

Before using this information, be sure to read the general information under “Notices” on page 85.

Compilation date: June 7, 2012

Contents

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an email form appears.
  3. Fill out the email form as instructed, and submit your feedback.
- To send comments on PDF books, you can email your comments to: **wasdoc@us.ibm.com**.

  Your comment should pertain to specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer. When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about your comments.

Using this PDF

Links

Because the content within this PDF is designed for an online information center deliverable, you might experience broken links. You can expect the following link behavior within this PDF:

- Links to Web addresses beginning with http:// work.
- Links that refer to specific page numbers within the same PDF book work.
- The remaining links will *not* work. You receive an error message when you click them.

Print sections directly from the information center navigation

PDF books are provided as a convenience format for easy printing, reading, and offline use. The information center is the official delivery format for IBM WebSphere Application Server documentation. If you use the PDF books primarily for convenient printing, it is now easier to print various parts of the information center as needed, quickly and directly from the information center navigation tree.

To print a section of the information center navigation:

1. Hover your cursor over an entry in the information center navigation until the **Open Quick Menu** icon is displayed beside the entry.
2. Right-click the icon to display a menu for printing or searching your selected section of the navigation tree.
3. If you select **Print this topic and subtopics** from the menu, the selected section is launched in a separate browser window as one HTML file. The HTML file includes each of the topics in the section, with a table of contents at the top.
4. Print the HTML file.

For performance reasons, the number of topics you can print at one time is limited. You are notified if your selection contains too many topics. If the current limit is too restrictive, use the feedback link to suggest a preferable limit. The feedback link is available at the end of most information center pages.

Chapter 1. Tuning the Liberty profile

Use this topic to learn about the tunable parameters and attributes of the Liberty profile.

About this task

The Liberty profile supports different attributes in the `server.xml` file to influence application performance. You can use these parameters and attributes to achieve better performance.

Procedure

- Tune the JVM.

  Tuning the JVM is a most important tuning step whether you are configuring a development or production environment. When tuning the JVM for the Liberty profile, consider using the `jvm.options` file in the `${server.config.dir}` directory. You can specify each of the JVM arguments that you want to use, one option per line. See Customizing the Liberty profile environment for more information. An example of `jvm.options` file as follows:

  ```
  -Xms50m
  -Xmx256m
  ```

  For a development environment, you might be interested in faster server startup, so consider setting the minimum heap size to a small value, and the maximum heap size to whatever value is needed for your application. For a production environment, setting the minimum heap size and maximum heap size to the same value can provide the best performance by avoiding heap expansion and contraction.

- Tune transport channel services.

  The transport channel services manage client connections, I/O processing for HTTP, thread pools, and connection pools. For applications on the Liberty profile, the following attributes are available for different elements that can be used to improve runtime performance, or scalability, or both. Each of these attributes is also described in Liberty profile: Configuration elements in the server.xml file.

  `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 can provide benefit to low latency or high throughput applications and SSL connections where building up new a connection can be costly. Here is an example of how you code this option in the `server.xml` file:

  ```
  <httpOptions maxKeepAliveRequests="-1" />
  ```

  `coreThreads` of `executor`

  This option specifies the core number of threads to associate with the executor of the thread pool. The number of threads associated with the executor will quickly grow to this number. 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 2-core SMT-4 machine, which represents 8 logical processors, you might use **`coreThreads`**`="40"` as a starting point.

  Here is an example of how you code this option in the `server.xml` file:

  ```
  <executor name="LargeThreadPool" id="default" coreThreads="40" maxThreads="80"
            keepAlive="60s" stealPolicy="STRICT" rejectedWorkPolicy="CALLER_RUNS" />
  ```

  `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 multicore (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 setting **numConnectionsPerThreadLocal** to 1 or more, that number of connections per thread are stored in thread local storage. When using **numConnectionsPerThreadLocal**, two other values need to be considered:

- The number of application threads
- The connection pool maximum connections

For best performance, if you have n applications threads, set maximum pool connections to at least n times the value of **numConnectionsPerThreadLocal** attribute. For example, if you use 20 application threads, then the maximum pool connections should be set to 20 or more; If you set the value of **numConnectionPerThreadLocal** attribute as 2 and there are 20 application threads, then the maximum pool connection must be set 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. This option must be set by reviewing the application code (or an SQL trace gathered from the database or database driver) for all unique prepared statements, and ensuring 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 datasource isolation level is used to specify the degree of data integrity and concurrency, which in turns controls the level of database locking. Traditionally there are four different options, listed below 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">
```

Chapter 2. Planning for performance

How well a website performs while receiving heavy user traffic is an essential factor in the overall success of an organization. This section provides online resources that you can consult to ensure that your site performs well under pressure.

Procedure

- Consult the following web resources for learning.

  IBM® Patterns for e-Business

  IBM Patterns for e-business is a group of reusable assets that can help speed the process of developing Web-based applications. The patterns leverage the experience of IBM architects to create solutions quickly, whether for a small local business or a large multinational enterprise.

  Planning for availability in the enterprise

  Availability is an achievable service-level characteristic that every enterprise struggles with. The worst case scenario is realized when load is underestimated or bandwidth is overloaded because availability planning was not carefully conducted. Applying the information in this article and the accompanying spreadsheet to your planning exercises can help you avoid such a scenario.

  Hardware configurations for WebSphere® Application Server production environments

  This article describes the most common production hardware configurations, and provides the reasons for choosing each one. It begins with a single machine configuration, and then proceeds with additional configurations that have higher fault tolerance, horizontal scaling, and a separation of web and enterprise bean servers.

- Take advantage of performance functions to improve performance. You can use functions such as balancing workloads with clusters and using the dynamic cache to improve performance.

Application design consideration

This topic describes architectural suggestions for the design and tuning of applications.

The designing applications information contains the architectural suggestions and the implementation of applications. For existing applications, the suggestions might require changing the existing implementations. Tuning the application server and resource parameters can have the greatest effect on performance of the applications that are well designed.

Use designing applications considerations in this topic for tips to ensure your applications are thoughtfully designed and tuned. These considerations include websites and other ideas for finding best practices for designing WebSphere applications, particularly in the realm of WebSphere extensions to the Java Platform, Enterprise Edition (Java EE) specification.

best-practices: Use the following information as an architectural guide when implementing applications:

- Persistence
- Model-view-controller pattern
- Statelessness
- Caching
- Asynchronous considerations
- Third-party libraries

Java EE applications load, store, create, and remove data from relational databases, a process commonly referred to as *persistence*. Most enterprise applications have significant database access. The architecture

and performance of the persistence layer is critical to the performance of an application. Therefore, persistence is a very important area to consider when making architectural choices that require trade-offs related to performance. This guide recommends first focusing on a solution that has clean architecture. The clean architecture considers data consistency, security, maintenance, portability, and the performance of that solution. Although this approach might not yield the absolute peak performance obtainable from manual coding a solution that ignores the mentioned qualities of service, this approach can achieve the appropriate balance of data consistency, maintainability, portability, security, and performance.

Multiple options are available in Java EE for persistence: Session beans using entity beans including container-managed persistence (CMP) or bean-managed persistence (BMP), session beans using Java Database Connectivity (JDBC), and Java beans using JDBC. For the reasons previously mentioned, consider CMP entity persistence because it provides maximum security, maintenance, and portability. CMP is also recommended for good performance. Refer to the Tune the EJB container section of the tuning application servers topic on tuning enterprise beans and more specifically, CMP.

If an application requires using enterprise beans not using EJB entities, the persistence mechanism usually involves the JDBC API. Because JDBC requires manual coding, the Structured Query Language (SQL) that runs against a database instance, it is critical to optimize the SQL statements that are used within the application. Also, configure the database server to support the optimal performance of these SQL statements. Finally, usage of specific JDBC APIs must be considered including prepared statements and batching.

Regardless of which persistence mechanism is considered, use container-managed transactions where the bean delegates management of transactions to the container. For applications that use JDBC, this is easily achieved by using the session façade pattern, which wraps all JDBC functions with a stateless session bean.

Finally, information about tuning the connection over which the EJB entity beans or JDBC communicates can be found in the Tune the data sources section of the tuning application servers topic.

One of the standard Java EE programming architectures is the model-view-controller (MVC) architecture, where a call to a controller servlet might include one or more child JavaServer Pages (JSP) files to construct the view. The MVC pattern is a recommended pattern for application architecture. This pattern requires distinct separation of the view (JSP files or presentation logic), the controller (servlets), and the model (business logic). Using the MVC pattern enables optimization of the performance and scalability of each layer separately.

Implementations that avoid storing the client user state scale and perform the best. Design implementations to avoid storing state. If state storage is needed, ensure that the size of the state data and the time that the state is stored are kept to the smallest possible values. Also, if state storage is needed, consider the possibility of reconstructing the state if a failure occurs, instead of guaranteeing state failover through replication.

Specific tuning of state affects HTTP session state, dynamic caching, and enterprise beans. Refer to the follow tuning guides for tuning the size, replication, and timing of the state storage:
- Session management tuning
- EJB tuning tips
- Tuning dynamic cache with the cache monitor

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 front-end 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.

Network designers must consider caching when network planning is performed because caching also integrates at most levels of the network topology. For applications that are available on the public Internet, network designers might want to consider Edge Side Include (ESI) caching when WebSphere Application Server caching extends into the public Internet. Network caching services are available in the proxy server for WebSphere Application Server, WebSphere Edge Component Caching Proxy, and the WebSphere plug-in.

Java EE workloads typically consist of two types of operations. You must perform the first type of operation to respond to a system request. You can perform the second type of operation asynchronously after the user request that initiated the operation is fulfilled.

An example of this difference is an application that enables you to submit a purchase order, enables you to continue while the system validates the order, queries remote systems, and in the future informs you of the purchase order status. This example can be implemented synchronously with the client waiting for the response. The synchronous implementation requires application server resources and you wait until the entire operations complete. If the process enables you to continue, while the result is computed asynchronously, the application server can schedule the processing to occur when it is optimal in relation to other requests. The notification to you can be triggered through email or some other interface within the application.

Because the asynchronous approach supports optimal scheduling of workloads and minimal server resource, consider asynchronous architectures. WebSphere Application Server supports asynchronous programming through Java EE Java Message Service (JMS) and message-driven beans (MDB) as well as asynchronous beans that are explained in the Tuning Java Message Service and Tuning MDB topics.

Verify that all the libraries that applications use are also designed for server-side performance. Some libraries are designed to work well within a client application and fail to consider server-side performance concerns, for example, memory utilization, synchronization, and pooling. It is suggested that all libraries that are not developed as part of an application undergo performance testing using the same test methodologies as used for the application.

Additional references:IBM WebSphere Developer Technical Journal: The top 10 Java EE best practicesImprove performance in your XML applications, Part 2

Chapter 3. Taking advantage of performance functions

This topic highlights a few main ways you can improve performance through a combination of product features and application development considerations.

Procedure

- Use one of the following considerations to improve performance.

  Balancing workloads with clusters

  Clusters are sets of servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine. A cell can have no clusters, one cluster, or multiple clusters.

  Using the dynamic cache service to improve performance

  The dynamic cache service improves performance by caching the output of servlets, commands, and JavaServer Pages (JSP) files. Dynamic caching features include cache replication among clusters, cache disk offload, Edge-side include caching, and external caching, which is the ability to control caches outside of the application server, such as that of your web server.

- Ensure your applications perform well.

  Take advantage of architectural suggestions and coding best practices to ensure that your applications perform well. See the information about application design considerations and the information on designing applications to learn more about ways you can improve performance of your applications.

Chapter 4. Obtaining advice from the advisors

Advisors provide a variety of recommendations that help improve the performance of your application server.

Before you begin

The advisors provide helpful performance as well as diagnostic advice about the state of the application server.

About this task

Tuning WebSphere Application Server is a critical part of getting the best performance from your website. However, 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.

The Runtime Performance Advisor is extended to also provide diagnostic advice and is now called the Performance and Diagnostic Advisor. Diagnostic advice provides useful information regarding the state of the application server. Diagnostic advice is especially useful when an application is not functioning as expected, or simply as a means of monitoring the health of application server.

Procedure

- Decide which performance advisor is right for the purpose, Performance and Diagnostic Advisor or Tivoli® Performance Viewer advisor.
- Use the chosen advisor to periodically check for inefficient settings, and to view recommendations.
- Analyze Performance Monitoring Infrastructure data with performance advisors.

Why you want to use the performance advisors

The advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere Application Server-specific rules for tuning. The advisors that are based on this information provide advice on how to set some of your configuration parameters to better tune WebSphere Application Server.

The advisors provide a variety of advice on the following application server resources:

- Object Request Broker service thread pools
- Web container thread pools
- Connection pool size
- Persisted session size and time
- Data source statement cache size
- Session cache size
- Dynamic cache size
- Java virtual machine heap size
- DB2® Performance Configuration wizard
- Connection use violations

For example, consider the data source statement cache. It optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not used in an active connection. (Both statements are SQL statements that essentially run repeatable tasks without the costs of repeated compilation.) If the cache is full, an old entry in the cache is discarded to make room for the new one. The best performance is generally obtained when the cache is large enough to hold all of the statements that are used in the application. The PMI counter, prepared statement cache discards, indicates the number of statements that are discarded from the cache. The performance advisors check this counter and provide recommendations to minimize the cache discards.

Another example is thread or connection pooling. The idea behind pooling is to use an existing thread or connection from the pool instead of creating a new instance for each request. Because each thread or connection in the pool consumes memory and increases the context-switching cost, the pool size is an important configuration parameter. A pool that is too large can hurt performance as much as a pool that is too small. The performance advisors use PMI information about current pool usage, minimum or maximum pool size, and the application server CPU utilization to recommend efficient values for the pool sizes.

The advisors can also issue diagnostic advice to help in problem determination and health monitoring. For example, if your application requires more memory than is available, the diagnostic adviser tells you to increase the size or the heap for application server.

Performance advisor types and purposes

Two performance advisors are available: the Performance and Diagnostic Advisor and the performance advisor in Tivoli Performance Viewer.

The Performance and Diagnostic Advisor runs in the Java virtual machine (JVM) process of application server; therefore, it does not provide expensive advice. In a stand-alone application server environment, the performance advisor in Tivoli Performance Viewer runs within the application server JVM.

The performance advisor in Tivoli Performance Viewer provides advice to help tune systems for optimal performance and provide recommendations on inefficient settings by using collected Performance Monitoring Infrastructure (PMI) data. Obtain the advice by selecting the performance advisor in Tivoli Performance Viewer.

In a WebSphere Application Server, Network Deployment environment, the performance advisor in Tivoli Performance Viewer runs within the JVM of the node agent and can provide advice on resources that are more expensive to monitor and analyze. The Tivoli Performance Viewer advisor requires that you enable performance modules, counters, or both.

Table 1. Performance and Diagnostic Advisor and Tivoli Performance Viewer advisor. The following chart shows the differences between the Performance and Diagnostic Advisor and the Tivoli Performance Viewer advisor:

	Performance and Diagnostic Advisor	**Tivoli Performance Viewer advisor**
Start location	Application server	Tivoli Performance Viewer client
Invocation of tool	Administrative console	Tivoli Performance Viewer
Output	• The `SystemOut.log` file • The administrative console • JMX notifications	Tivoli Performance Viewer in the administrative console
Frequency of operation	Configurable	When you select refresh in the Tivoli Performance Viewer administrative console

Table 1. Performance and Diagnostic Advisor and Tivoli Performance Viewer advisor (continued). The following chart shows the differences between the Performance and Diagnostic Advisor and the Tivoli Performance Viewer advisor:

	Performance and Diagnostic Advisor	**Tivoli Performance Viewer advisor**
Types of advice	Performance advice: • Object Request Broker (ORB) service thread pools • Web container thread pools • Connection pool size • Persisted session size and time • Prepared statement cache size • Session cache size • Memory leak detection Diagnostic advice: • Connection factory diagnostics • Data source diagnostics Connection usage diagnostics • Detection of connection use by multiple threads • Detection of connection use across components	Performance advice: • ORB service thread pools • Web container thread pools • Connection pool size • Persisted session size and time • Prepared statement cache size • Session cache size • Dynamic cache size • Java virtual machine (JVM) heap size • DB2 Performance Configuration wizard

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Performance and Diagnostic Advisor

Use this topic to understand the functions of the Performance and Diagnostic Advisor.

The Performance and Diagnostic Advisor provides advice to help tune systems for optimal performance and is configured using the WebSphere Application Server administrative console or the wsadmin tool. Running in the Java virtual machine (JVM) of the application server, the Performance and Diagnostic Advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed both as warnings in the administrative console under Runtime Messages in the WebSphere Application Server Status panel and as text in the application server `SystemOut.log` file. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

The Performance and Diagnostic Advisor provides performance advice and diagnostic advice to help tune systems for optimal performance, and also to help understand the health of the system. It is configured

using the WebSphere Application Server administrative console or the wsadmin tool. Running in the Java virtual machine (JVM) of the application server, the Performance and Diagnostic Advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed as warnings in the administrative console under Runtime Messages in the WebSphere Application Server Status panel, as text in the application server `SystemOut.log` file, and as Java Management Extensions (JMX) notifications. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

From WebSphere Application Server, Version 6.0.2, you can use the Performance and Diagnostic Advisor to enable the lightweight memory leak detection, which is designed to provide early detection of memory problems in test and production environments.

The advice that the Performance and Diagnostic Advisor gives is all on the server level. The only difference when running in a WebSphere Application Server, Network Deployment environment is that you might receive contradictory advice on resources that are declared at the node or cell level and used at the server level.

For example, two sets of advice are given if a data source is declared at the node level to have a connection pool size of {10,50} and is used by two servers (server1 and server2). If server1 uses only two connections and server2 uses all fifty connections during peak load, the optimal connection pool size is different for the two servers. Therefore, the Performance and Diagnostic Advisor gives two sets of advice (one for server1 and another for server2). The data source is declared at the node level and you must make your decisions appropriately by setting one size that works for both, or by declaring two different data sources for each server with the appropriate level.

Read the using the performance and diagnostic advisor information for startup and configuration steps.

Diagnostic alerts:

In WebSphere Application Server Version 8.5 the Performance and Diagnostic Advisors are extended to provide more diagnostic alerts to help common troubleshoot problems.

Several alerts are made available to monitor connection factory and data sources behavior. Some of these alerts are straightforward and easy to comprehend. Others are much more involved and are intended for use by IBM support only.

ConnectionErrorOccured diagnostic alert

When a resource adapter or data source encounters a problem with connections such that the connection might no longer be usable, it informs the connection manager that a connection error occurred. This causes the destruction of the individual connection or a pool purge, which is the destruction of all connections in the pool, depending on the pool purge policy configuration setting. An alert is sent, indicating a potential problem with the back-end if an abnormally high number of unusable connections are detected.

Connection low-percent efficiency diagnostic alert

If the percentage of time that a connection is used versus held for any individual connections drops beneath a threshold, an alert is sent with a call stack.

Cross-Component Use JCA Programming Model Violation Diagnostic Alert

When you enable cross-component use detection, the application server raises an alert when a connection handle is used by a Java EE application component that is different from the component that originally acquired the handle through a connection factory. This condition might inadvertently occur if an application passes a connection handle in a parameter or an application obtains a handle from a cache that is shared

by multiple application components. If components use a connection handle in this manner, this might result in problems with application or data integrity. Enable the alert to detect the cross-component connection use during development to identify and avoid potential application problems.

Local transaction containment (LTC) nesting threshold exceeded diagnostic alert

For LTC definition, see the Local transaction containment (LTC) and Transaction type and connection behavior information, and Default behavior of managed connections in WebSphere Application Server topic.

If a high number of LTCs are started on a thread before completing, an alert is raised. This alert is useful in debugging some situations where the connection pool is unexpectedly running out of connections due to multiple nested LTCs holding onto multiple shareable connections.

Multi-Thread Use JCA Programming Model Violation Diagnostic Alert

Multi-thread use detection raises an alert when an application component acquires a connection handle using a connection factory, and then the component uses the handle on a different thread from which the handle was acquired. If you use a connection in this manner, this behavior might cause data integrity problems, especially if an application uses a connection handle on a thread that is not managed. Enable the alert to detect multi-thread connection usage during application development.

Pool low-percent efficiency diagnostic alert

If the average time that a connection is held versus used for the all connections in the pool drops beneath a threshold, an alert is sent.

Serial reuse violation diagnostic alert

For information on what serial reuse is, see the transaction type and connection behavior information. Some legitimate scenarios exist, where a serial reuse violation is appropriate, but in most cases this violation is not intended and might lead to data integrity problems.

If this alert is enabled, any time a serial reuse violation occurs within an LTC, an alert is sent.

Surge mode entered or exited diagnostic alert

When surge mode is configured, an alert is sent whenever surge mode engages or disengages. See the surge mode documentation for more information.

Stuck connection block mode entered or exited diagnostic alert

When stuck connection detection is configured, an alert is sent whenever stuck connection blocking starts or stops. See the stuck connection information.

Thread maximum connections exceeded diagnostic alert

When one or more LTCs on a thread ties too many managed connections, or poolable connections for data sources an alert is issued.

Using the Performance and Diagnostic Advisor

The advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere Application Server-specific rules for tuning.

AIX Linux Windows

About this task

This topic is only appropriate for AIX®, Linux, and Windows operating systems.

The Performance and Diagnostic Advisor provides advice to help tune systems for optimal performance and is configured using the WebSphere Application Server administrative console or the wsadmin tool (scripting). The Performance and Diagnostic Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning. Running in the Java virtual machine (JVM) of the application server, this advisor periodically checks for inefficient settings, and issues recommendations as standard product warning messages. View these recommendations by clicking **Troubleshooting > Runtime Messages > Runtime Warning** in the administrative console. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

Procedure

1. Ensure that PMI is enabled, which is default. If PMI is disabled, see the enabling PMI using the administrative console information. To obtain advice, you must first enable PMI through the administrative console and restart the server. The Performance and Diagnostic Advisor enables the appropriate monitoring counter levels for all enabled advice when PMI is enabled. If specific counters exist that are not wanted, or when disabling the Performance and Diagnostic Advisor, you might want to disable PMI or the counters that the Performance and Diagnostic Advisor enabled.
2. If running WebSphere Application Server, Network Deployment, you must enable PMI on both the server and the administrative agent, and restart the server and the administrative agent.
3. Click **Servers** > **Application servers** in the administrative console navigation tree.
4. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.
5. Under the **Configuration** tab, specify the number of processors on the server. This setting is critical to ensure accurate advice for the specific configuration of the system.
6. Select the **Calculation Interval**. PMI data is taken over time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Therefore, details within the advice messages display as averages over this interval.
7. Select the **Maximum Warning Sequence**. The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is updated. For example, if the maximum warning sequence is set to `3`, then the advisor sends only three warnings, to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is issued only if the rate of discards exceeds the new threshold setting.
8. Specify **Minimum CPU for Working System**. The minimum central processing unit (CPU) for a working system refers to the CPU level that indicates a application server is under production load. Or, if you want to tune your application server for peak production loads that range from 50-90% CPU utilization, set this value to `50`. If the CPU is below this value, some diagnostic and performance advice are still issued. For example, regardless of the CPU level if you are discarding prepared statements at a high rate, you are notified.
9. Specify **CPU Saturated**. The CPU saturated level indicates at what level the CPU is considered fully utilized. The level determines when concurrency rules no longer increase thread pools or other resources, even if they are fully utilized.
10. Click **Apply**.
11. Click **Save**.
12. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.

13. Click the **Runtime** tab.
14. Click **Restart**. Select **Restart** on the Runtime tab to reinitialize the Performance and Diagnostic Advisor using the last configuration information that is saved to disk.

    This action also resets the state of the Performance and Diagnostic Advisor. For example, the current warning count is reset to zero (0) for each message.
15. Simulate a production level load. If you use the Performance and Diagnostic Advisor in a test environment, do any other tuning for performance, or simulate a realistic production load for your application. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory, to the levels that are expected in production. The Performance and Diagnostic Advisor provides advice when CPU utilization exceeds a sufficiently high level only. For a list of IBM business partners that provide tools to drive this type of load, see the performance: resource for learning information.
16. Select the check box to enable the Performance and Diagnostic Advisor.

    Tip: To achieve the best results for performance tuning, enable the Performance and Diagnostic Advisor when a stable production-level load is applied.
17. Click **OK**.
18. Select **Runtime Warnings** in the administrative console under the Runtime Messages in the Status panel or look in the `SystemOut.log` file, which is located in the following directory:

    profile_root`/logs/`*server_name*

    Some messages are not issued immediately.
19. Update the product configuration for improved performance, based on advice. Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure that it functions and performs better than the previous configuration.

    Over time, the advisor might issue differing advice. The differing advice is due to load fluctuations and the runtime state. When differing advice is received, you need to look at all advice and the time period over which it is issued. Advice is taken during the time that most closely represents the peak production load.

    Performance tuning is an iterative process. After applying advice, simulate a production load, update the configuration that is based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

What to do next

You can enable and disable advice in the Advice Configuration panel. Some advice applies only to certain configurations, and can be enabled only for those configurations. For example, unbounded Object Request Broker (ORB) service thread pool advice is only relevant when the ORB service thread pool is unbounded, and can only be enabled when the ORB thread pool is unbounded. For more information on Advice configuration, see the advice configuration settings information.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Performance and Diagnostic Advisor configuration settings

Use this page to specify settings for the Performance and Diagnostic Advisor.

To view this administrative page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name*** > **Performance and Diagnostic Advisor Configuration** under the Performance section.

Enable Performance and Diagnostic Advisor Framework

Specifies whether the Performance and Diagnostic Advisor runs on the server startup.

The Performance and Diagnostic Advisor requires that the Performance Monitoring Infrastructure (PMI) be enabled. It does not require that individual counters be enabled. When a counter that is needed by the Performance and Diagnostic Advisor or is not enabled, the Performance and Diagnostic Advisor enables it automatically. When disabling the Performance and Diagnostic Advisor, you might want to disable Performance Monitoring Infrastructure (PMI) or the counters that Performance and Diagnostic Advisor enabled. The following counters might be enabled by the Performance and Diagnostic Advisor:

- ThreadPools (module)
  - Web Container (module)
    - Pool Size
    - Active Threads
  - Object Request Broker (module)
    - Pool Size
    - Active Threads
- JDBC Connection Pools (module)
  - Pool Size
  - Percent used
  - Prepared Statement Discards
- Servlet Session Manager (module)
  - External Read Size
  - External Write Size
  - External Read Time
  - External Write Time
  - No Room For New Session
- System Data (module)
  - CPU Utilization
  - Free Memory

Enable automatic heap dump collection

Specifies whether the Performance and Diagnostic Advisor automatically generates heap dumps for post analysis when suspicious memory activity is detected.

Calculation Interval

Specifies the length of time over which data is taken for this advice.

PMI data is taken over an interval of time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Details within the advice messages display as averages over this interval. The default value is automatically set to four minutes.

Maximum warning sequence

The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is relaxed.

For example, if the maximum warning sequence is set to 3, the advisor only sends three warnings to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is only issued if the rate of discards exceeds the new threshold setting. The default value is automatically set to one.

Number of processors
Specifies the number of processors on the server.

This setting is helpful to ensure accurate advice for the specific configuration of the system. Depending your configuration and system, there may be only one processor utilized. The default value is automatically set to two.

Minimum CPU For Working System
The minimum CPU for working system refers to the point at which concurrency rules do not attempt to free resources in thread pools.

There is a set of concurrency alerts to warn you if all threads in a pool are busy. This can affect performance, and it may be necessary for you to increase them. The CPU bounds are a mechanism to help determine when an application server is active and tunable.

The Minimum CPU for working system sets a lower limit as to when you should consider adjusting thread pools. For example, say you set this value to 50%. If the CPU is less than 50%, concurrency rules *do not* try to free up resources by decreasing pools to get rid of unused threads. That is, if the pool size is 50-100 and only 20 threads are consistently used then concurrency rules would like to decrease the minimum pool size to 20.

CPU Saturated
The CPU Saturated setting determines when the CPU is deemed to be saturated.

There is a set of concurrency alerts to warn you if all threads in a pool are busy. This can affect performance, and it may be necessary for you to increase them. The CPU bounds are a mechanism to help determine when an application server is active and tunable.

The CPU saturated setting determines when the CPU has reached its saturation point. For example, if this is set to 95%, when the CPU is greater than 95% the concurrency rules *do not* try to improve things, that is, increase the size of a thread pool.

Advice configuration settings
Use this page to select the advice you wish to enable or disable.

To view this administrative page, click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***. Under the Performance section, click **Performance and Diagnostic Advisor Configuration** > **Performance and Diagnostic Advice Configuration**.

Advice name
Specifies the advice that you can enable or disable.

Advice applied to component
Specifies the WebSphere Application Server component to which the advice applies.

Advice type
Categorizes the primary indent of a piece of Advice.

Use Advice type for grouping, and then enabling or disabling sets of advice that is based upon your purpose. Advice has the following types:

- Performance: Performance advice provides tuning recommendations, or identifies problems with your configuration from a performance perspective.
- Diagnostic: Diagnostic advice provide automated logic and analysis relating to problem identification and analysis. These types advice are usually issued when unexpected circumstances are encountered by the application server.

Performance impact
Generalizes the performance overhead that an alert might incur.

The performance impact of a particular piece of advice is highly dependant upon the scenario being run and upon the conditions meet. The performance categorization of alerts is based upon worst case scenario measurements. The performance categorizations are:

- **Low:** Advice has minimal performance overhead. Advice might be run in test and production environments. Cumulative performance overhead is within run to run variance when all advice of this type is enabled.
- **Medium:** Advice has measurable but low performance overhead. Advice might be run within test environments, and might be run within production environments if deemed necessary. Cumulative performance overhead is less than 4% when all advice of this type is enabled.
- **High:** Advice impact is high or unknown. Advice might be run during problem determination tests and functional tests. It is not run in production simulation or production environments unless deemed necessary. Cumulative performance overhead might be significant when all advice of this type is enabled.

Advice status
Specifies whether the advice is stopped, started, or unavailable.

The advice status has one of three values: **Started**, **Stopped** or **Unavailable**.

- **Started**: The advice is enabled.
- **Stopped**: The advice is not enabled.
- **Unavailable**: The advice does not apply to the current configuration, for example, persisted session size advice in a configuration without persistent sessions.

Viewing the Performance and Diagnostic Advisor recommendations
Runtime Performance Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning.

About this task

The Performance and Diagnostic Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning. Running in the Java virtual machine (JVM) of the application server, this advisor periodically checks for inefficient settings, and issues recommendations as standard product warning messages.

Procedure

The Performance and Diagnostic Advisor recommendations are displayed in two locations:

1. The WebSphere Application Server `SystemOut.log` log file.
2. The Runtime Messages panel in the administrative console. To view this administrative page, click **Troubleshooting** > **Runtime Messages** > **Runtime Warning**.

Example

The following log file is a sample output of advice on the `SystemOut.log` file:

```
[4/2/04 15:50:26:406 EST] 6a83e321 TraceResponse W CWTUN0202W:
Increasing the web container thread pool Maximum Size to 48
might improve performance.

Additional explanatory data follows.

Average number of threads: 48.
```

```
Configured maximum pool size: 2.

This alert has been issued 1 time(s) in a row.
The threshold will be updated to reduce the
overhead of the analysis.
```

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Starting the lightweight memory leak detection

Use this task to start the lightweight memory leak detection using the Performance and Diagnostic Advisor.

Before you begin

If you have a memory leak and want to confirm the leak, or you want to automatically generate heap dumps on Java virtual machines (JVM) in WebSphere Application Server, consider changing your minimum and maximum heap sizes to be equal. This change provides the memory leak detection more time for reliable diagnosis.

About this task

To start the lightweight memory leak detection using the Performance and Diagnostic Advisor, perform the following steps in the administrative console:

Procedure

1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* **> Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Enable the Performance and Diagnostic Advisor Framework.
5. Click **OK**.
6. From the Runtime or Configuration tab of Performance and Diagnostic Advisor Framework, click **Performance and Diagnostic Advice configuration**.
7. Start the memory leak detection advice and stop any other unwanted advice.

Results

The memory leak detection advice is started.

Important: To achieve the best results for performance tuning, start the Performance and Diagnostic Advisor when a stable production level load is running.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

What to do next

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the “Viewing the Performance and Diagnostic Advisor recommendations” on page 18 topic.

Lightweight memory leak detection

This topic describes memory leaks in Java applications and introduces lightweight memory leak detection.

Although a Java application has a built-in garbage collection mechanism, which frees the programmer from any explicit object deallocation responsibilities, memory leaks are still common in Java applications. Memory leaks occur in Java applications when unintentional references are made to unused objects. This occurrence prevents Java garbage collection from freeing memory.

The term *memory leak* is overused; a memory leak refers to a memory misuse or mismanagement. Old unused data structures might have outstanding references but are never garbage collected. A data structure might have unbounded growth or there might not be enough memory that is allocated to efficiently run a set of applications.

Most existing memory leak technologies are based upon the idea that you know that you have a memory leak and want to find it. Because of these analysis requirements, these technologies have significant performance burdens and are not designed for use as a detection mechanism in production. This limitation means that memory leaks are generally not detected until the problem is critical; the application passes all system tests and is put in production, but it crashes and nobody knows why.

WebSphere Application Server has implemented a lightweight memory leak detection mechanism that runs within the WebSphere Performance and Diagnostic Advisor framework. This mechanism is designed to provide early detection of memory problems in test and production environments. This framework is not designed to provide analysis of the source of the problem, but rather to provide notification and help generating the information that is required to use analysis tools. The mechanism only detects memory leaks in the Java heap and does not detect native leaks.

The lightweight memory leak detection in WebSphere Application Server does not require any additional agents. The detection relies on algorithms that are based on information that is available from the Performance Monitoring Infrastructure service and has minimal performance overhead.

Enabling automated heap dump generation

Use this task to enable automated heap dump generation. This function is not supported when using a Sun Java virtual machine (JVM) which includes WebSphere Application Server running on HP-UX and Solaris operating systems. You need to research taking heap dumps on Sun JVMs or call IBM Support.

Before you begin

Although heap dumps are only generated in response to a detected memory leak, you must understand that generating heap dumps can have a severe performance impact on WebSphere Application Server for several minutes.

About this task

The automated heap dump generation support, which is available only on IBM Software Development Kit and analyzes memory leak problems on AIX, Linux, and Windows operating systems.

Manually generating heap dumps at appropriate times might be difficult. To help you analyze memory leak problems when memory leak detection occurs, some automated heap dump generation support is available. This functionality is available only for IBM Software Development Kit on AIX, Linux, and Windows operating systems.

Most memory leak analysis tools perform some forms of difference evaluation on two heap dumps. Upon detection of a suspicious memory situation, two heap dumps are automatically generated at appropriate times. The general idea is to take an initial heap dump as soon as problem detection occurs. Monitor the memory usage and take another heap dump when you determine that enough memory is leaked, so that you can compare the heap dumps to find the source of the leak.

To help you analyze memory leak problems when memory leak detection occurs, some automated heap dump generation support is available.

To enable automated heap dump generation support, perform the following steps in the administrative console:

Procedure

1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* >**Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Select the **Enable automatic heap dump collection** check box.
5. Click **OK**.

Results

The automated heap dump generation support is enabled.

Important: To preserve disk space, the Performance and Diagnostic Advisor does not take heap dumps if more than 10 heap dumps already exist in the WebSphere Application Server home directory. Depending on the size of the heap and the workload on the application server, taking a heap dump might be quite expensive and might temporarily affect system performance.

The automatic heap dump generation process dynamically reacts to various memory conditions and generates dumps only when it is needed. When the heap memory is too low, the heap dumps cannot be taken or the heap dump generation cannot be complete.

What to do next

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the “Viewing the Performance and Diagnostic Advisor recommendations” on page 18 topic. If a memory leak is detected and you want to find the heap dump, refer to the Locating and analyzing heap dumps topic.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Generating heap dumps manually

Use this task to generate heap dumps manually. This function is not supported on when using a Sun Java virtual machine (JVM) which includes WebSphere Application Server running on HP-UX and Solaris operating systems.

Before you begin

Windows AIX Linux Although heap dumps are generated only in response to a detected memory leak, you must understand that generating heap dumps can have a severe performance impact on WebSphere Application Server for several minutes. When generating multiple heap dumps manually for memory leak analysis, make sure that significant objects are leaked in between the two heap dumps. This approach enables problem determination tools to identify the source of the memory leak.

About this task

You might want to manually generate heap dumps for the analysis of memory leaks. You might also want to designate certain times to take heap dumps because of the overhead involved. On JVM in WebSphere Application Server, you can manually produce heap dumps by using the generateHeapDump operation on WebSphere Application Server managed beans (MBeans) that are special Java beans.

Solaris HP-UX On a Java virtual machines (JVM) in WebSphere Application Server, you cannot enable automated heap dump generation.

Procedure

1. Determine if you want to use wsadmin or the administrative console to generate your heap dump.
2. To use wsadmin to generate your heap dump, complete the following:
   a. Start the wsadmin scripting client. You have several options to run scripting commands, ranging from running them interactively to running them in a profile.
   b. Invoke the generateHeapDump operation on a JVM MBean, for example:
      - Find a JVM objectName:

        ```
        <wsadmin> set objectName [$AdminControl queryNames
        WebSphere:type=JVM,process=<servername>,node=<nodename>,*]
        ```

      - Invoke the generateHeapDump operation on JVM MBean:

        ```
        <wsadmin> $AdminControl invoke $objectName generateHeapDump
        ```

Table 2. Description of variables. The following table explains variables in the command previously mentioned.

Variable	Description
`$`	is a Jacl operator for substituting a variable name with its value
`invoke`	is the command
`generateHeapDump`	is the operation you are invoking
<servername>	is the name of the server on which you want to generate a heap dump
<nodename>	is the node to which *<servername>* belongs

3. To use the administrative console to generate your heap dump, complete the following:
   a. Start the administrative console.
   b. In the navigation pane, click **Troubleshooting** > **Java dumps and cores**.
   c. Select the *server_name* for which you want to generate the heap dump.
   d. Click **Heap dump** to generate the heap dump for your specified server.

What to do next

After running the **wsadmin** command, the file name of the heap dump is returned. For more information on finding heap dumps, refer to the Locating and analyzing heap dumps topic. When you have a couple of heap dumps, use a number of memory leak problem determination tools to analyze your problem. Memory Dump Diagnostic for Java™ is an offline tool for diagnosing root causes behind memory leaks in the Java

heap. See the diagnosing out-of-memory errors and Java heap memory leaks information.

Locating and analyzing heap dumps

Use this task to locate and analyze heap dumps.

Before you begin

Do not analyze heap dumps on the WebSphere Application Server machine because analysis is very expensive. For analysis, transfer heap dumps to a dedicated problem determination machine.

About this task

When a memory leak is detected and heap dumps are generated, you must analyze heap dumps on a problem determination machine and not on the application server because the analysis is very central processing unit (CPU) and disk I/O intensive.

Perform the following procedure to locate the heap dump files.

Procedure

1. On the physical application server where a memory leak is detected, go to the WebSphere Application Server home directory. For example, on the Windows operating system, the directory is:

   profile_root\myProfile
2. IBM heap dump files are usually named in the following way:

   heapdump.<*date*>.<*timestamp*><*pid*>.phd
3. Gather all the .phd files and transfer them to your problem determination machine for analysis.
4. Many tools are available to analyze heap dumps that include Rational® Application Developer 6.0. WebSphere Application Server serviceability released a technology preview called Memory Dump Diagnostic For Java. You can download this preview from the product download website.

What to do next

When you have a couple of heap dumps, use a number of memory leak problem determination tools to analyze your problem.

Using the performance advisor in Tivoli Performance Viewer

The performance advisor in Tivoli Performance Viewer provides advice to help tune systems for optimal performance and provides recommendations on inefficient settings by using the collected Performance Monitoring Infrastructure (PMI) data.

About this task

Obtain advice by clicking **Performance Advisor** in Tivoli Performance Viewer. The performance advisor in Tivoli Performance Viewer provides more extensive advice than the Performance and Diagnostic Advisor. For example, Tivoli Performance Viewer provides advice on setting the dynamic cache size, setting the Java virtual machine (JVM) heap size and using the DB2 Performance Configuration wizard.

Procedure

1. Enable PMI in the application server.

   To monitor performance data through the PMI interfaces, you must first enable PMI through the administrative console before restarting the server.

   If running in a WebSphere Application Server, Network Deployment environment, you must enable PMI on both the server and on the administrative agent before restarting the server and the administrative agent.

2. Enable data collection and set the PMI monitoring level to Extended.

   The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server. These monitoring levels and the data selected determine the type of advice you obtain. The performance advisor in Tivoli Performance Viewer uses the extended monitoring level; however, the performance advisor in Tivoli Performance Viewer can use a few of the more expensive counters (to provide additional advice) and provide advice on which counters can be enabled.

   For example, the advice pertaining to session size needs the PMI statistic set to All. Or, you can use the PMI Custom Monitoring Level to enable the Servlet Session Manager SessionObjectSize counter. The monitoring of the SessionSize PMI counter is expensive, and is not in the Extended PMI statistic set. Complete this action in one of the following ways:

   a. PMI settings.

   b. Enabling Performance Monitoring Infrastructure using the wsadmin tool.

3. In the administrative console, click **Monitoring and Tuning** > **Performance Viewer** > **Current Activity**.
4. Simulate a production level load. Simulate a realistic production load for your application, if you use the performance advisor in a test environment, or do any other performance tuning. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory to the levels that are expected in production. The performance advisor only provides advice when CPU utilization exceeds a sufficiently high level. For a list of IBM business partners providing tools to drive this type of load, see the performance: resource for learning information.
5. Log performance data with Tivoli Performance Viewer.
6. Clicking **Refresh** on top of the table of advice causes the advisor to recalculate the advice based on the current data in the buffer.
7. Tuning advice displays when the Advisor icon is chosen in the Tivoli Performance Viewer Performance Advisor. Double-click an individual message for details. Because PMI data is taken over an interval of time and averaged to provide advice, details within the advice message display as averages.

   Note: If the Refresh Rate is adjusted, the Buffer Size must also be adjusted to enable sufficient data to be collected for performing average calculations. Currently 5 minutes of data is required. Hence, the following guidelines intend to help you use the Tivoli Performance Advisor:

   a. You cannot have a Refresh Rate of more than 300 seconds.

   b. RefreshRate * BufferSize > 300 seconds. Buffer Size * Refresh Rate is the amount of PMI data available in memory and it must be greater than 300 seconds.

   c. For the Tivoli Performance Advisor to work properly with Tivoli Performance Viewer logs, the logs must be at least 300 seconds of duration.

   For more information about configuring user and logging settings of Tivoli Performance Viewer, refer to the configuring Tivoli Performance Viewer settings information.
8. Update the product configuration for improved performance, based on advice. Because Tivoli Performance Viewer refreshes advice at a single instant in time, take the advice from the peak load time. Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs well.

   Over a period of time the advisor might issue differing advice. The differing advice is due to load fluctuations and run-time state. When differing advice is received, you need to look at all advice and the time period over which it was issued. You must take advice during the time that most closely represents the peak production load.

Performance tuning is an iterative process. After applying advice, simulate a production load, update the configuration that is based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

Performance advisor report in Tivoli Performance Viewer

View recommendations and data from the performance advisor in Tivoli Performance Viewer by clicking the Advisor link in Tivoli Performance Viewer for a server.

For more information on how to use the performance advisor in Tivoli Performance Viewer, see the article, Using the performance advisor in Tivoli Performance Viewer.

Message

Specifies recommendations for performance tuning.

Click the message to obtain more details.

Performance data in the upper panel

Displays a summary of performance data for WebSphere Application Server. Data here corresponds to the same period that recommendations were provided for. However, recommendations might use a different set of data points during analysis than the set that is displayed by the summary page.

The first table represents the number of requests per second and the response time in milliseconds for both the web and Enterprise JavaBeans containers.

The pie graph displays the CPU activity as percentage busy and idle.

The second table displays the average thread activity for the web container and Object Request Broker (ORB) thread pools, and the average database connection activity for connection pools. The activity is expressed as the number of threads or connections busy and idle.

Chapter 5. Tuning the application serving environment

Use this topic to understand the benefits of tuning for optimal performance. Learn about the tunable parameters of the major WebSphere Application Server components and how these parameters affect performance.

Before you begin

WebSphere Application Server provides tunable settings for its major components so that you can adjust the runtime environment to match the characteristics of your application. Applications can run successfully without changing the default values for these tuning parameters. Other applications might need changes, for example, a larger heap size, to achieve optimal performance.

Performance tuning can yield significant gains in performance even if an application is not optimized for performance. However, correcting shortcomings of an application typically results in higher performance gains than are possible with just altering tuning parameters. Many factors contribute to a high performing application.

About this task

Procedure

1. Run the **applyPerfTuningTemplate.py** script as the starting point for improving the performance of a specific application server. This python-based tuning script, along with one of its template files, applies the recommended performance tuning settings for a typical development, production, or environment that is ready for immediate use. The **applyPerfTuningTemplate.py** script, and its associated templates and properties files, are located in the *WAS_HOME*/bin directory.
2. Use the performance advisors, the suggested procedures or parameters in the tuning parameter hot list, and the information on troubleshooting performance problems to optimize your WebSphere Application Server instances to their fullest extent.

   Performance advisors
   The performance advisors use the Performance Monitoring Infrastructure (PMI) data to suggest configuration changes to Object Request Broker (ORB) service thread pools, web container thread pools, connection pool size, persisted session size and time, prepared statement cache size, and session cache size. The Runtime Performance Advisor runs in the application server process, while the other advisor runs in the Tivoli Performance Viewer. For more information, see the documentation about using the Performance and Diagnostic Advisor and use the performance advisor in Tivoli Performance Viewer.

   Tuning parameter hot list
   Review the documentation about the tuning parameter hot list. These parameters have an important impact on performance. Because these parameters are application-dependent, the parameter settings for specific applications and environments can vary.

   Troubleshooting performance
   To save you time detecting problems and help you troubleshoot performance problems, see the documentation about troubleshooting performance.

Tuning parameter hot list

The following hot list contains recommendations that have improved performance or scalability, or both, for many applications.

WebSphere Application Server provides several tunable parameters and options to match the application server environment to the requirements of your application.

- Review the hardware and software requirements

It is critical for proper functionality and performance to satisfy the minimum hardware and software requirements. Refer to IBM WebSphere Application Server supported hardware, software, and APIs website which details hardware and software requirements.

- Install the most current refresh pack, fix pack, and the recommended interim fixes

  The list of recommended updates is maintained on the Support site.
- Check hardware configuration and settings

  Verify network interconnections and hardware configuration is setup for peak performance.
- Tune the operating systems

  Operating system configuration plays a key role in performance. For example, adjustments such as TCP/IP parameters might be necessary for your application
- Set the minimum and maximum Java virtual machine (JVM) heap sizes

  Many applications need a larger heap size then the default for best performance. It is also advised to select an appropriate GC policy based on the application's characteristics.
- Use a type 4 (or pure Java) JDBC driver

  In general, the type 2 JDBC driver is recommended if the database exists on the same physical machine as the WebSphere instance. However, in the case where the database is in a different tier, the type 4 JDBC driver offers the fastest performance since they are pure Java not requiring native implementation. Use the previous link to view a list of database vendor-specific requirements, which can tell you if a type 4 JDBC driver is supported for your database.

  See the *Administering applications and their environment* PDF for more information.
- Tune WebSphere Application Server JDBC data sources and associated connection pools

  The JDBC data source configuration might have a significant performance impact. For example, the connection pool size and prepared statement cache need to be sized based on the number of concurrent requests being processed and the design of the application.

  See the *Administering applications and their environment* PDF for more information.
- Enable the pass by reference option

  Use applications that can take advantage of the pass by reference option to avoid the cost of copying parameters to the stack.
- Ensure that the transaction log is assigned to a fast disk

  Some applications generate a high rate of writes to the WebSphere Application Server transaction log. Locating the transaction log on a fast disk or disk array can improve response time

  See the *Administering applications and their environment* PDF for more information.
- Tune related components, for example, database

  In many cases, some other component, for example, a database, needs adjustments to achieve higher throughput for your entire configuration.

  For more information, see the *Administering applications and their environment* PDF for more information.
- Disable functions that are not required

  For example, if your application does not use the web services addressing (WS-Addressing) support, disabling this function can improve performance.

  Attention: Use this property with care because applications might require WS-Addressing MAPs to function correctly. Setting this property also disables WebSphere Application Server support for the following specifications, which depend on the WS-Addressing support: Web Services Atomic Transactions, Web Services Business Agreement and Web Services Notification.

  To disable the support for WS-Addressing, refer to Enabling Web Services Addressing support for JAX-RPC applications
- Review your application design

  You can track many performance problems back to the application design. Review the design to determine if it causes performance problems.

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations (distributed)

The following file paths are default locations. You can install the product and other components or create profiles in any directory where you have write access. Multiple installations of WebSphere Application Server Network Deployment products or components require multiple locations. Default values for installation actions by root and nonroot users are given. If no nonroot values are specified, then the default directory values are applicable to both root and nonroot users.

app_client_root

Table 3. Default installation root directories for the Application Client for IBM WebSphere Application Server.

This table shows the default installation root directories for the Application Client for IBM WebSphere Application Server.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppClient` (Java EE Application client only) HP-UX Linux Solaris `/opt/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\Program Files\IBM\WebSphere\AppClient`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\IBM\WebSphere\AppClient`

app_server_root

Table 4. Default installation directories for WebSphere Application Server.

This table shows the default installation directories for WebSphere Application Server Network Deployment.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppServer` HP-UX Linux Solaris `/opt/IBM/WebSphere/AppServer` Windows `C:\Program Files\IBM\WebSphere\AppServer`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppServer` Windows *user_home*`\IBM\WebSphere\AppServer`

component_root

The component installation root directory is any installation root directory described in this article. Some programs are for use across multiple components—in particular, the Web Server Plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

gskit_root

IBM Global Security Kit (GSKit) can now be installed by any user. GSKit is installed locally inside the installing product's directory structure and is no longer installed in a global location on the target system.

Table 5. Default installation directories for GSKit.

This table shows the default installation root directory for Version 8 of the GSKit, where product_root is the root directory of the product that is installing GSKit, for example IBM HTTP Server or the web server plug-in.

User	Directory
Root and nonroot	AIX HP-UX Linux Solaris *product_root*/gsk8 Windows *product_root*\gsk8

profile_root

Table 6. Default profile directories.

This table shows the default directories for a profile named profile_name on each distributed operating system.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer/profiles/*profile_name* HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer/profiles/*profile_name* Windows C:\Program Files\IBM\WebSphere\AppServer\profiles*profile_name*
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/profiles Windows *user_home*\IBM\WebSphere\AppServer\profiles

plugins_root

Table 7. Default installation root directories for the Web Server Plug-ins.

This table shows the default installation root directories for the Web Server Plug-ins for WebSphere Application Server.

User	Directory
Root	AIX /usr/IBM/WebSphere/Plugins HP-UX Linux Solaris /opt/IBM/WebSphere/Plugins Windows C:\Program Files\IBM\WebSphere\Plugins
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Plugins Windows C:\IBM\WebSphere\Plugins

wct_root

Table 8. Default installation root directories for the WebSphere Customization Toolbox.

This table shows the default installation root directories for the WebSphere Customization Toolbox.

User	Directory
Root	AIX /usr/IBM/WebSphere/Toolbox HP-UX Linux Solaris /opt/IBM/WebSphere/Toolbox Windows C:\Program Files\IBM\WebSphere\Toolbox
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Toolbox Windows C:\IBM\WebSphere\Toolbox

web_server_root

Table 9. Default installation root directories for the IBM HTTP Server.

This table shows the default installation root directories for the IBM HTTP Server.

User	Directory
Root	AIX /usr/IBM/HTTPServer HP-UX Linux Solaris /opt/IBM/HTTPServer Windows C:\Program Files\IBM\HTTPServer
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/HTTPServer Windows C:\IBM\HTTPServer

Tuning TCP/IP buffer sizes

WebSphere Application Server uses the TCP/IP sockets communication mechanism extensively. For a TCP/IP socket connection, the send and receive buffer sizes define the receive window. The receive window specifies the amount of data that can be sent and not received before the send is interrupted. If too much data is sent, it overruns the buffer and interrupts the transfer. The mechanism that controls data transfer interruptions is referred to as flow control. If the receive window size for TCP/IP buffers is too small, the receive window buffer is frequently overrun, and the flow control mechanism stops the data transfer until the receive buffer is empty.

About this task

Flow control can consume a significant amount of CPU time and result in additional network latency as a result of data transfer interruptions. It is recommended that you increase buffer sizes to avoid flow control under normal operating conditions. A larger buffer size reduces the potential for flow control to occur, and results in improved CPU utilization. However, a large buffer size can have a negative effect on performance in some cases. If the TCP/IP buffers are too large and applications are not processing data fast enough, paging can increase. The goal is to specify a value large enough to avoid flow control, but not so large that the buffer accumulates more data than the system can process.

The default buffer size is 8 KB. The maximum size is 8 MB (8096 KB). The optimal buffer size depends on several network environment factors including types of switches and systems, acknowledgment timing, error rates and network topology, memory size, and data transfer size. When data transfer size is extremely large, you might want to set the buffer sizes up to the maximum value to improve throughput, reduce the occurrence of flow control, and reduce CPU cost.

Buffer sizes for the socket connections between the web server and WebSphere Application Server are set at 64KB. In most cases this value is adequate.

Flow control can be an issue when an application uses either the IBM Developer Kit for Java(TM) JDBC driver or the IBM Toolbox for Java JDBC driver to access a remote database. If the data transfers are large, flow control can consume a large amount of CPU time. If you use the IBM Toolbox for Java JDBC driver, you can use custom properties to configure the buffer sizes for each data source. It is recommended that you specify large buffer sizes, for example, 1 MB.

Some system-wide settings can override the default 8 KB buffer size for sockets. With some applications, for example, WebSphere Commerce Suite, a buffer size of 180 KB reduces flow control and typically does not adversely affect paging. The optimal value is dependent on specific system characteristics. You might need to try several values before you determine the ideal buffer size for your system.

AIX For more information, see “TCP/IP network settings ”in Running IBM WebSphere Application Server on System p and AIX: Optimization and Best Practices . In addition, see TCP streaming workload tuning.

Linux For more information, see the following documents:

- Boost socket performance on Linux
- Linux Tuning

Solaris For more information, see “Network configuration for WebSphere Application Server communication ”in IBM WebSphere Application Server V6.1 on the Solaris 10 Operating System

Windows For information on tuning TCP/IP buffer sizes, see the Windows 2000 and Windows Server 2003 TCP Featuresdocument. Consider setting the TcpWindowSize value to either 8388608 or 16777216.

Tuning the JVM

Tuning the IBM virtual machine for Java

An application server is a Java based server and requires a Java virtual machine (JVM) environment to run and support the enterprise applications that run on it. As part of configuring your application server, you can configure the Java SE Runtime Environment to tune performance and system resource usage. This topic applies to IBM virtual machines for Java.

Before you begin

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

- Determine the type of JVM on which your application server is running.

  Issue the **`java –fullversion`** command from within your application server *app_server_root*`/java/bin` directory.

  In response to this command, Java writes information about the JVM, including the JVM provider information, into window where you run the command; for example:

  ```
  java full version "JRE 1.6.0 IBM Windows 32 build pwi3260sr7-20091217_01 (SR7)"
  ```

  If your application server is running on a Sun HotSpot JVM, see the topic *Tuning Sun HotSpot Java virtual machines (Solaris & HP-UX)*.

- Verify that:
  1. The most recent supported version of the JVM is installed on your system.
  2. The most recent service update is installed on your system. Almost every new service level includes JVM performance improvements.

About this task

Each JVM vendor provides detailed information on performance and tuning for their JVM. Use the information provided in this topic in conjunction with the information that is provided with the JVM that is running on your system.

A Java SE Runtime Environment provides the environment for running enterprise applications and application servers. Therefore the Java configuration plays a significant role in determining performance and system resource consumption for an application server and the applications that run on it.

The IBM virtual machine for Java Version 6.0 includes the latest in Java Platform, Enterprise Edition (Java EE) specifications, and provides performance and stability improvements over previous versions of Java.

Even though JVM tuning is dependent on the JVM provider you use, there are some general tuning concepts that apply to all JVMs. These general concepts include:

- Compiler tuning. All JVMs use Just-In-Time (JIT) compilers to compile Java byte codes into native instructions during server runtime.
- Java memory or heap tuning. Tuning the JVM memory management function, or garbage collection, is a good starting point for improving JVM performance.
- Class loading tuning.
- Start up versus runtime performance optimization

The following steps provide specific instructions on how to perform the following types of tuning for each JVM. The steps do not have to be performed in any specific order.

Procedure

1. Optimize the startup and runtime performance.

   In some environments, such as a development environment, it is more important to optimize the startup performance of your application server rather than the runtime performance. In other environments, it is more important to optimize the runtime performance. By default, IBM virtual machines for Java are optimized for runtime performance, while HotSpot-based JVMs are optimized for startup performance.

   The Java Just-in-Time (JIT) compiler impacts whether startup or runtime performance is optimized. The initial optimization level that the compiler uses influences the length of time that is required to compile a class method, and the length of time that is required to start the server. For faster startups, reduce the initial optimization level that the compiler uses. However if you reduce the initial optimization level, the runtime performance of your applications might decrease because the class methods are now compiled at a lower optimization level.

   - **-Xquickstart**

     This setting influences how the IBM virtual machine for Java uses a lower optimization level for class method compiles. A lower optimization level provides for faster server startups, but lowers runtime performance. If this parameter is not specified, the IBM virtual machine for Java defaults to starting with a high initial optimization level for compiles, which results in faster runtime performance, but slower server starts.

     You can set this property on the Java virtual machine panel using the administrative console. For details, read the information about Java virtual machine settings.

Information	**Value**
Default	High initial compiler optimization level
Recommended	High initial compiler optimization level
Usage	Specifying `-Xquickstart` improves server startup time.

2. Configure the heap size.

   The Java heap parameters influence the behavior of garbage collection. Increasing the heap size supports more object creation. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact and causes garbage collection to take longer.

   The JVM uses defined thresholds to manage the storage that it is allocated. When the thresholds are reached, the garbage collector is invoked to free up unused storage. Therefore, garbage collection can cause significant degradation of Java performance. Before changing the initial and maximum heap sizes, you should consider the following information:

- In the majority of cases you should set the maximum JVM heap size to a value that is higher than the initial JVM heap size. This setting allows for the JVM to operate efficiently during normal, steady state periods within the confines of the initial heap. This setting also allows the JVM to operate effectively during periods of high transaction volume because the JVM can expand the heap up to the value specified for the maximum JVM heap size. In some rare cases, where absolute optimal performance is required, you might want to specify the same value for both the initial and maximum heap size. This setting eliminates some overhead that occurs when the JVM expands or contracts the size of the JVM heap. Before changing any of the JVM heap sizes, verify that the JVM storage allocation is large enough to accommodate the new heap size.
- Do not make the size of the initial heap so large that while it initially improves performance by delaying garbage collection, when garbage collection does occur, the collection process affects response time because the process has to run longer.

To use the administrative console to configure the heap size:

a. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.

b. In the Server Infrastructure section, click **Java and process management > Process definition > Java virtual machine**.

c. Specify a new value in either the **Initial heap size** or the **Maximum heap size** field.

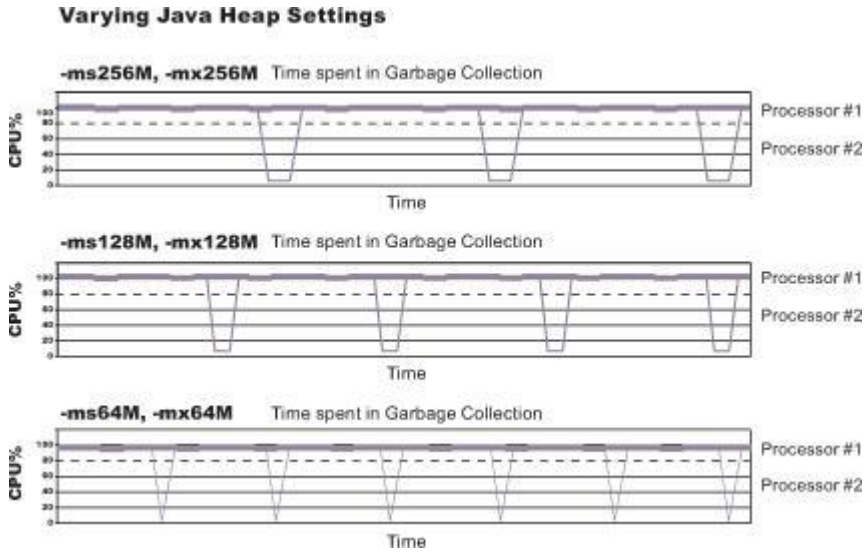
   You can also specify values for both fields if you need to adjust both settings.

   For performance analysis, the initial and maximum heap sizes should be equal.

   The Initial heap size setting specifies, in megabytes, the amount of storage that is allocated for the JVM heap when the JVM starts. The Maximum heap size setting specifies, in megabytes, the maximum amount of storage that can be allocated to the JVM heap. Both of these settings have a significant effect on performance.

   If you are tuning a production system where you do not know the working set size of the enterprise applications that are running on that system, an appropriate starting value for the initial heap size is 25 percent of the maximum heap size. The JVM then tries to adapt the size of the heap to the working set size of the application.

   The following illustration represents three CPU profiles, each running a fixed workload with varying Java heap settings. In the middle profile, the initial and maximum heap sizes are set to 128 MB. Four garbage collections occur. The total time in garbage collection is about 15 percent of the total run. When the heap parameters are doubled to 256 MB, as in the top profile, the length of the work time increases between garbage collections. Only three garbage collections occur, but the length of each garbage collection is also increased. In the third profile, the heap size is reduced to 64 MB and exhibits the opposite effect. With a smaller heap size, both the time between garbage collections and the time for each garbage collection are shorter. For all three configurations, the total time in garbage collection is approximately 15 percent. This example illustrates an important concept about the Java heap and its relationship to object utilization. A cost for garbage collection always exists when running enterprise applications.



Run a series of tests that vary the Java heap settings. For example, run experiments with 128 MB, 192 MB, 256 MB, and 320 MB. During each experiment, monitor the total memory usage. If you expand the heap too aggressively, paging can occur.

Use the **vmstat** command or the Windows Performance Monitor to check for paging. If paging occurs, reduce the size of the heap or add more memory to the system.

When all the runs are finished, compare the following statistics:

- Number of garbage collection calls
- Average duration of a single garbage collection call
- Ratio between the length of a single garbage collection call and the average time between calls

If the application is not over utilizing objects and has no memory leaks, the state of steady memory utilization is reached. Garbage collection also occurs less frequently and for short duration.

If the heap free space settles at 85 percent or more, consider decreasing the maximum heap size values because the application server and the application are under-utilizing the memory allocated for heap.

d. Click **Apply**.

e. Click **Save** to save your changes to the master configuration.

f. Stop and restart the application server.

You can also use the following command-line parameters to adjust these settings. These parameters apply to all supported JVMs and are used to adjust the minimum and maximum heap size for each application server or application server instance.

- **-Xms**

  This parameter controls the initial size of the Java heap. Tuning this parameter reduces the overhead of garbage collection, which improves server response time and throughput. For some applications, the default setting for this option might be too low, which causes a high number of minor garbage collections.

Information	Value
Default	50 MB
Recommended	Workload specific, but higher than the default.
Usage	Specifying `-Xms256m` sets the initial heap size to 256 MB.

- **-Xmx**

  This parameter controls the maximum size of the Java heap. Increasing this parameter increases the memory available to the application server, and reduces the frequency of garbage collection.

Increasing this setting can improve server response time and throughput. However, increasing this setting also increases the duration of a garbage collection when it does occur. This setting should never be increased above the system memory available for the application server instance. Increasing the setting above the available system memory can cause system paging and a significant decrease in performance.

Information	Value
Default	By default, the JVM dynamically calculates the Java heap size based on the available memory in the system.
Recommended	Workload specific, but higher than the default value, depending on the amount of available physical memory.
Usage	Specifying `-Xmx512m` sets the maximum heap size to 512 MB.

Note: Specify a value for the **-Xmx** parameter to reduce possible out-of-memory issues.

- AIX Windows **-Xlp**

  Use this parameter with the IBM virtual machine for Java to allocate the heap when using large pages, such as 16 MB pages. Before specifying this parameter, verify that your operating system is configured to support large pages. Using large pages can reduce the CPU overhead needed to keep track of heap memory, and might also allow the creation of a larger heap.

Default
64 KB if you are using Java 6 SR 7 or higher 4 KB if you are using Java 6 SR 6 or lower

- AIX **–Xlp64k**

  This parameter can be used to allocate the heap using medium size pages, such as 64 KB. Using this virtual memory page size for the memory that an application requires can improve the performance and throughput of the application because of hardware efficiencies that are associated with a larger page size.

  AIX i5/OS® and AIX provide rich support around 64 KB pages because 64 KB pages are intended to be general purpose pages. 64 KB pages are easy to enable, and applications might receive performance benefits when 64 KB pages are used. Starting with Java 6 SR 7, the Java heap is allocated with 64K pages by default. For Java 6 SR 6 or earlier, 4K pages is the default setting, This setting can be changed without changing the operating system configuration. However, it is recommended that you run your application servers in a separate storage pool if you use of 64KB pages.

  AIX If you are using Java 6 SR 6 or earlier, to support a 64 KB page size, in the administrative console, click **Servers** > **Application servers** > *server_name* > **Process definition** > **Environment entries** > **New**, and then specify `LDR_CNTRL` in the **Name** field and `DATAPSIZE=64K@TEXTPSIZE=64K@STACKPSIZE=64K` in the **Value** field.

Recommended
Use 64 KB page size whenever possible. AIX POWER5+ systems, and AIX 5L™ Version 5.3 with the 5300-04 Recommended Maintenance Package support a 64 KB page size when they are running the 64-bit kernel.

- AIX **–Xlp4k**

This parameter can be used to allocate the heap using 4 KB pages. Using this virtual memory page size for the memory that an application requires, instead of 64 KB, might negatively impact performance and throughput of the application because of hardware inefficiencies that are associated with a smaller page size.

AIX Starting with Java 6 SR 7, the Java heap is allocated with 64K pages by default. For Java 6 SR 6 or earlier, 4K pages is the default setting, This setting can be changed without changing the operating system configuration. However, it is recommended that you run your application servers in a separate storage pool if you use of 64KB pages.

AIX If you are using Java 6 SR 7 or higher, to support a 4 KB page size, in the administrative console, click **Servers > Application servers >** *server_name* **> Process definition > Environment entries > New**, and then specify `LDR_CNTRL` in the **Name** field and `DATAPSIZE=4K@TEXTPSIZE=4K@STACKPSIZE=4K` in the **Value** field.

Recommended
Use -Xlp64k instead of -Xlp4k whenever possible.

3. Tune Java memory.

   Enterprise applications written in the Java language involve complex object relationships and use large numbers of objects. Although, the Java language automatically manages memory associated with object life cycles, understanding the application usage patterns for objects is important. In particular, verify that the following conditions exist:
   - The application is not over utilizing objects
   - The application is not leaking objects
   - The Java heap parameters are set properly to handle a given object usage pattern

   a. Check for over-utilization of objects.

      You can review the counters for the JVM run time, that are included in Tivoli Performance Viewer reports, to determine if an application is overusing objects. You have to specify the **`-XrunpmiJvmtiProfiler`** command-line option, as well as the JVM module maximum level, to enable the Java virtual machine profiler interface, JVMTI, counters.

      The optimal result for the average time between garbage collections is at least five to six times the average duration of a single garbage collection. If you do not achieve this number, the application is spending more than 15 percent of its time in garbage collection.

      If the information indicates a garbage collection bottleneck, there are two ways to clear the bottleneck. The most cost-effective way to optimize the application is to implement object caches and pools. Use a Java profiler to determine which objects to target. If you can not optimize the application, try adding memory, processors and clones. Additional memory allows each clone to maintain a reasonable heap size. Additional processors allow the clones to run in parallel.

   b. Test for 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:

- **Long-running test**

  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.

You can use the Tivoli Performance Viewer to help find memory leaks.

For optimal results, repeat experiments with increasing duration, such as 1,000, 2,000, and 4,000 page requests. The Tivoli Performance Viewer graph of used memory should have a jagged shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following conditions is appears in the graph:

- The amount of memory used immediately after each garbage collection increases significantly. When this condition occurs, the jagged pattern looks more like a staircase.
- The jagged pattern has an irregular shape.
- The gap between the number of objects allocated and the number of objects freed increases over time.

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.

Heap fragmentation can be reduced by forcing compactions to occur. However, there is a performance penalty for forcing compactions. Use the Java **-X** command to see the list of memory options.

4. Tune garbage collection

   Examining Java garbage collection gives insight to how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. Garbage collection typically consumes from 5 to 20 percent of total run time of a properly functioning application. If not managed, garbage collection is one of the biggest bottlenecks for an application.

   Monitoring garbage collection while a fixed workload is running, provides you with insight as to whether the application is over using objects. Garbage collection can even detect the presence of memory leaks.

   You can use JVM settings to configure the type and behavior of garbage collection. When the JVM cannot allocate an object from the current heap because of lack of contiguous space, the garbage collector is invoked to reclaim memory from Java objects that are no longer being used. Each JVM vendor provides unique garbage collector policies and tuning parameters.

   You can use the **Verbose garbage collection** setting in the administrative console to enable garbage collection monitoring. The output from this setting includes class garbage collection statistics. The format of the generated report is not standardized between different JVMs or release levels.

   To adjust your JVM garbage collection settings:

   a. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.
   b. In the Server Infrastructure section, click **Java and process management > Process definition > Java virtual machine**
   c. Enter the –X option you want to change in the **Generic JVM arguments** field.
   d. Click **Apply**.
   e. Click **Save** to save your changes to the master configuration.
   f. Stop and restart the application server.

   The following list describes the –X options for the different JVM garbage collectors.

   The IBM virtual machine for Java garbage collector.

   A complete guide to the IBM implementation of the Java garbage collector is provided in the *IBM Developer Kit and Runtime Environment, Java2 Technology Edition, Version 5.0 Diagnostics Guide*. This document is available on the developerWorks® website.

   Use the Java **-X** option to view a list of memory options.

   - **-Xgcpolicy**

     The IBM virtual machine for Java provides four policies for garbage collection. Each policy provides unique benefits.

Note: While each policy provides unique benefits, for WebSphere Application Server Version 8.0 and later, gencon is the default garbage collection policy. Previous versions of the application server specify that optthruput is the default garbage collection policy.

– gencon is the default policy. This policy works with the generational garbage collector. The generational scheme attempts to achieve high throughput along with reduced garbage collection pause times. To accomplish this goal, the heap is split into new and old segments. Long lived objects are promoted to the old space while short-lived objects are garbage collected quickly in the new space. The gencon policy provides significant benefits for many applications. However, it is not suited for all applications, and is typically more difficult to tune.
– optthruput provides high throughput but with longer garbage collection pause times. During a garbage collection, all application threads are stopped for mark, sweep and compaction, when compaction is needed. The gencon policy is sufficient for most applications.
– optavgpause is the policy that reduces garbage collection pause time by performing the mark and sweep phases of garbage collection while an application is running. This policy causes a small performance impact to overall throughput.
– subpool is a policy that increases performance on multiprocessor systems, that commonly use more then 8 processors. This policy is only available on IBM System i® System p® and System z® processors. The subpool policy is similar to the gencon policy except that the heap is divided into subpools that provide improved scalability for object allocation.

Information	Value
Default	gencon
Recommended	gencon
Usage	Specifying `Xgcpolicy:gencon` sets the garbage collection policy to gencon.

Setting **gcpolicy** to `gencon` disables concurrent mark. You should get optimal throughput results when you use the gencon policy unless you are experiencing erratic application response times, which is an indication that you might have pause time problems

Setting **gcpolicy** to `optavgpause` enables concurrent mark with its default values. This setting alleviates erratic application response times that normal garbage collection causes. However, this option might decrease overall throughput.

- **-Xnoclassgc**

  By default, the JVM unloads a class from memory whenever there are no live instances of that class left. The overhead of loading and unloading the same class multiple times, can decrease performance.

  gotcha: You can use the `-Xnoclassgc` argument to disable class garbage collection. However, the performance impact of class garbage collection is typically minimal, and turning off class garbage collection in a Java Platform, Enterprise Edition (Java EE) based system, with its heavy use of application class loaders, might effectively create a memory leak of class data, and cause the JVM to throw an Out-of-Memory Exception.

  If you use this option, whenever you redeploy an application, you should always restart the application server to clear the classes and static data from the pervious version of the application.

Information	Value
Default	Class garbage collection is enabled.
Recommended	Do not disable class garbage collection.
Usage	Specify `Xnoclassgc` to disable class garbage collection.

5. **Enable localhost name caching** By default in the IBM SDK for Java, the static method java/net/InetAddress.getLocalHost does not cache its result. This method is used throughout WebSphere Application Server, but particularly in administrative agents such as the deployment manager and node agent. If the localhost address of a process will not change while it is running, then it is advised to use a built-in cache for the localhost lookup by setting the com.ibm.cacheLocalHost system property to the value true. Refer to the Java virtual machine custom properties topic in the information center for instructions on setting JVM custom properties on the various types of processes.

   Note: The address for servers configured using DHCP change over time. Do not set this property unless you are using statically assigned IP addresses for your server.

Information	Value
Default	com.ibm.cacheLocalHost = *false*
Recommended	com.ibm.cacheLocalHost = *true* (see description)
Usage	Specifying `-Dcom.ibm.cacheLocalhost=true` enables the getLocalHost cache

6. **Enable class sharing in a cache.**

   The share classes option of the IBM implementation of the Java 2 Runtime Environment (J2RE) Version 1.5.0 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.

   gotcha: Solaris HP-UX The IBM implementation of J2RE Version 1.5.0 is currently not supported on:

   - Solaris Solaris
   - HP-UX HP-UX

   If you use this option, you should clear the cache when the process is not in use. To clear the cache, either call the *app_server_root*`/bin/clearClassCache.bat/sh` utility or stop the process and then restart the process.

   If you need to disable the share classes option for a process, specify the generic JVM argument -Xshareclasses:none for that process:

   a. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.
   b. In the Server Infrastructure section, click **Java and process management > Process definition > Java virtual machine**
   c. Enter `-Xshareclasses:none` in the **Generic JVM arguments** field.
   d. Click **OK**.
   e. Click **Save** to save your changes to the master configuration.
   f. Stop and restart the application server.

Information	Value
Default	The Share classes in a cache option are enabled.
Recommended	Leave the share classes in a cache option enabled.

Information	Value
Usage	Specifying `-Xshareclasses:none` disables the share classes in a cache option.

7. Enable compressed references on 64-bit environments.

   You can enable compressed references on 64-bit environments, such as AIX 64, Linux PPC 64, zLinux 64, and Microsoft Windows AMD64, Linux AMD64.

   The compressed references option of the IBM implementation of the 64-bit Java SE Runtime Environment (JRE) Version 6.0 lets you limit all of the memory references to 32-bit size. Typically, the 64-bit JVMs use more heap space than the 32-bit JVMs because they use 64-bit wide memory references to address memory. The heap that is addressable by the 64-bit reference is orders of magnitude larger than the 32-bit heap, but in the real world, a heap that requires all 64-bits for addressing is typically not required. Compressing the references reduces the size of the addresses and makes more efficient use of the heap. Compressing these references also improves the processor cache and bus utilization, thereby improving performance.

   gotcha:

   The compressed references feature is not supported on:

   - HP-UX 64-bit JVM
   - iSeries® Classic 64-bit JVM

   The product automatically enables pointer compression on the supported platforms by default if the heap size (controlled by the -Xmx parameter) is set under a certain heap size (around 25 GB depending on platform), else it will default to non-compressed references. The user can override these defaults by using the command line options below.

   The following command-line options control compressed references feature:

   -Xcompressedrefs
   This command-line option enables the compressed references feature. When the JVM is launched with this command line option it would use 32-bit wide memory references to address the heap. This feature can be used up to a certain heap size (around 29GB depending on the platform), controlled by -Xmx parameter.

   -Xnocompressedrefs
   This command-line options explicitly disable the compressed references feature. When the JVM is launches with this command line option it will use full 64-bit wide memory references to address the heap. This option can be used by the user to override the default enablement of pointer compression, if needed.

8. Tune the configuration update process for a large cell configuration.

   In a large cell configuration, you might need to determine whether configuration update performance or consistency checking is more important. The deployment manager maintains a master configuration repository for the entire cell. By default, when the configuration changes, the product compares the configuration in the workspace with the master repository to maintain workspace consistency. However, the consistency verification process can cause an increase in the amount of time to save a configuration change or to deploy a large number of applications. The following factors influence how much time is required:

   - The more application servers or clusters that are defined in a cell, the longer it takes to save a configuration change.
   - The more applications that are deployed in a cell, the longer it takes to save a configuration change.

   If the amount of time required to change a configuration change is unsatisfactory, you can add the `config_consistency_check` custom property to your JVM settings and set the value of this property to false.

   a. In the administrative console, click **System administration > Deployment manager**.

b. Under Server Infrastructure, select Java and Process Management, and then click **Process Definition**.
c. Under Additional Properties, click **Java Virtual Machine > Custom Properties > New**.
d. Enter `config_consistency_check` in the Name field and `false` in the Value field.
e. Click **OK** and then save these changes to the master configuration.
f. Restart the server.

Note: The `config_consistency_check` custom property affects the deployment manager process only. It does not affect other processes including the node agent and application server processes. The consistency check is not performed on these processes. However, within the `SystemOut.log` files for these processes, you might see a note that the consistency check is disabled. For these non-deployment manager processes, you can ignore this message.

If you are using the wsadmin command **`wsadmin -conntype none`** in local mode, you must set the **config_consistency_check** property to `false` before issuing this command.

What to do next

Continue to gather and analyze data as you make tuning changes until you are satisfied with how the JVM is performing.

Tuning HotSpot Java virtual machines (Solaris & HP-UX)

The architecture of the Sun-developed, HP-ported HotSpot Java virtual machine (JVM) has evolved differently than the IBM-developed software development kit (SDK.) Its internal structure, for young or old generation and permanent regions, arises to primarily support generational garbage collection, as well as other garbage collection modes as necessary.

Before you begin

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

- Determine the type of JVM on which your application server is running.

  Issue the **`java –fullversion`** command from within your application server *app_server_root*`/java/bin` directory. In response to this command, the application server writes information about the JVM, including the JVM provider information, into the `SystemOut.log` file. If your application server is running on an IBM virtual machine for Java, see the topic*Tuning the IBM virtual machine for Java*.
- Verify that the following statements are true for your system:
  1. The most recent supported version of the JVM is installed on your system.
  2. The most recent service update is installed on your system. Almost every new service level includes JVM performance improvements.

About this task

Tuning the Sun HotSpot JVM is an iterative process where the JVM configuration is developed, data gathered, primarily from verbosegc data, and then analyzed, and any configuration revisions applied on the next cycle. Perform one or more of the following steps if you need to tune your Sun HotSpot JVM.

Procedure

- Provide enough Java Heap Memory.

The Java heap memory is a reserved, contiguous set of addresses. The size of the Java heap memory is the maximum size for which the Java heap is configured. These addresses are not available for other native or system memory demands, and are maintained and managed only by the JVM because the Java heap is used for Java object storage for the lifetime of that JVM.

When the JVM initializes, secures resources for the Java heap are obtained according to the JVM configuration settings. If insufficient memory is available, the JVM initialization fails. If inadequate memory is configured in the Java heap, the system eventually fails with an OutOfMemory report, that is typically preceded by significant garbage collection activity, during which almost no Java processing occurs.

Sufficient consideration for the native memory needs of other components of your process must be made to accommodate running threads, storing data for I/O, and satisfying such requirements as alignment, and page size.

The Sun HotSpot Java heap comprises two physically independent parts that you must take into consideration when you specify maximum Java heap sizes:

– The permanent region, which is a combination of young and old generation regions that are further (subdivided into eden, survivor spaces, and tenured regions.
– The provision memory for the Java components of this system.

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.

Consult these tuning parameters:

– -XX:MaxPermSize (Permanent region)
– -Xmx (Maximum Java Heap size)

- Disable explicit garbage collection to eliminate any unnecessary or mistimed major garbage collection cycles that might be introduced in software components of the system.

  Consult the tuning parameter **-XX:+DisableExplicitGC.**

  gotcha: By default, the JVM unloads a class from memory whenever there are no live instances of that class left. You can use the `-Xnoclassgc` argument to disable class garbage collection. However, the performance impact of class garbage collection is typically minimal, and turning off class garbage collection in a Java Platform, Enterprise Edition (Java EE) based system, with its heavy use of application class loaders, might effectively create a memory leak of class data, and cause the JVM to throw an Out-of-Memory Exception.

  If you use the `-Xnoclassgc` argument, whenever you have to redeploy an application, you should always restart the application server to clear the classes and static data from the pervious version of the application.

  If you use the `-Xnoclassgc` argument, whenever you have to redeploy an application, you should always restart the application server to clear the classes and static data from the pervious version of the application.

- Tune region sizes to optimize garbage collection action.

  Any garbage collection tuning endeavour decisions should be based on the behavior of the garbage collectors. You should identify the correct garbage collection mode to suit the operational needs of you application. You should also verify that you are meeting your performance requirements, and are efficiently recycling enough memory resources to consistently meet the demands of your application.

Any changes that you make to garbage collection parameter settings should produce sufficiently different results and show benefits that are derived from exploiting different regions of the HotSpot Java heap.

An unwise choice typically lengthens the tuning process as the iterative tuning process needs to be substantially repeated. Further sections present the two principal choices, parallel throughput or concurrent low-pause, and the relevant options for further tuning. Both modes offer the potential for high performance, but the key performance factor is that the behavior that gets optimized is different for each mode.

The dominant tuning activity concerns controlling resource utilization to service allocation activity of the application, and to arrange efficient garbage collection to recycle storage, as required. Inevitably, these tuning discussions are dependent on the garbage collection mode employed. Two types of garbage collection are discussed:

– 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.
– A concurrent low-pause collector.

The objective of tuning with these collectors is to deliver the behavior that is most suited for the allocation patterns and object lifetimes of your application system, and that maximizes the efficiency of their collection actions.

– 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=

– Option 3: Use the concurrent low-pause mark-sweep collector

This collector is a radical departure from the evolution of generational garbage collection that has under pinned 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
- -XX:MaxTenuringThreshold=8
- -XX:NewSize=128m

Among the difficulties for tuning with CMS, is that the worst case garbage collection times, which is when the CMS cycle aborts, can take last 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.

gotcha: As with the throughput collector, there are considerably more options available for explicitly controlling CMS. However, those mentioned represent the core of the options that you are likely to need to considered using when you are tuning the HotSpot JVM.

What to do next

Gather and analyze data to evaluate the configuration, typically using verbosegc. Continue to gather and analyze data as you make tuning changes until you are satisfied with how the JVM is performing.

Sun HotSpot JVM tuning parameters (Solaris and HP-UX)

Tuning a Sun HotSpot Java virtual machine (JVM) is an iterative process where the JVM configuration is developed, data is gathered, primarily the verbosegc data, and then analyzed. Any configuration revisions are then applied on the next cycle. Even though there are many Sun HotSpot JVM parameters, the following parameters have been identified as central to tuning. Which of these parameters you modify depends on your configuration choices. Therefore, in addition to reviewing these parameter descriptions, it is strongly recommended that your read the topic “Tuning Sun HotSpot Java Virtual Machines (Solaris & HP-UX)” for a complete understanding or the JVM tuning methodology.

There is a standard form by which all Sun HotSpot options are specified. Understanding this form can help you avoid problems with transcribing options, interpreting instructions, and avoiding the potential confusion caused by the JVM rejecting an option, and then refusing to start.

You should be particularly concerned with the Sun HotSpot options that are particular to the implementation of the Sun HotSpot JVM, starting with the `-XX` option, rather than to standard or portable VM options, such as `-X` *option* or `-` *option*. The majority of these options are boolean valued, meaning they are set to either true or false. These settings either enable or disable a feature. The following standard form is used to enable an option, which is what you typically do when you change the setting of an option during the tuning process:

`-XX:+` *option*

The following standard form is used to disable an option, which you will do less frequently:

`-XX:-` *option*

gotcha:

- The use of a plus sign or a minus sign should immediately follows the colon. Otherwise the option typically requires a value, and appears more like the assignment of a value because it has an *option*=*value* format.
- As stated on the SUN website, the -XX Hotspot options are subject to change without notice in subsequent releases of the JDK. Therefore, before specifying an option, you should verify that it is supported for the version of the JDK that you are running on your system.

When determining which option to use, the name of the option typically describes the action that occurs if the option is enabled. The default value for most options leave the feature disabled. Therefore, if you disable an option that is already disabling a feature, it is possible to cause a double-negative situation. This is particularly true with options that have names that begins with the word Disable. For example, the default setting for the DisableExplicitGC option causes JVM to honor Explicit garbage collection requests. Therefore, you would normally want to enable this option by specifying a plus sign in front of this option. The plus sign has the affect of disabling the honoring of explicit garbage collection requests, which is what the name of the options implies. With options, such as the DisableExplicitGC option, it is rare to encounter the setting -XX:-DisableExplicitGC because this setting equates to specifying the default action.

In circumstances where the name of the option includes the term Use, the option typically makes more sense either for the enablement or disablement of that feature and the sense of the plus or minus sign is usually more intuitive.

Where a value needs to be specified, the option appears like an assignment with an equal sign between the option and the setting. In this situation, the option expects an appropriate number value to immediately follow the equal sign, without any blank spaces between the equal sign and the number. The value can often accept standard abbreviations, such as k for kilobytes, m for megabytes, and g for gigabytes, where it is appropriate to specify these values. The virtual machine performs only limited validation of such parameters, and, where invalid, typically produces an error message that indicates that the virtual machine cannot start.

-Xmx (Maximum Java Heap size):

Tune this parameter, in conjunction with the **`-XX:MaxPermSize`** parameter, to provide enough Java heap memory. When you specify a value for the maximum Java heap size for object storage in the Java heap, you should consider that the peak resource demands that are necessary for processing the peak input volumes that are designed to be handled by the system.

By contrast, the initial minimum size of the Java heap, that is specified using the **`-Xms`** parameter, should reflect the sizing of the Java heap that is needed to accommodate the persistent data that arises from the normal operation of the system under a routine steady-state input load. Such a resource request ensures an efficient system startup, where just the right amount of storage is claimed to permit quick initialization without needing many garbage collection cycles to increase heap capacity. Thereafter, the working size capacity of the Java heap varies between the normal capacity known to accommodate a routine

steady-state workload, and the systems design peak size, and any variations in heap capacity should reflect changes in the systems inputs, such as a burst of activity, or the increase of workload.

The working size capacity of the Java heap is considered useful information about the running state of the system. Tuning the initial minimum size of the Java heap should only involve optimizing system startup. Setting minimum and maximum heap sizes to the same value fixes the Java heap and constrains the recovery options of the JVM for its housekeeping of the Java heap. This type of setup can cause performance penalties and poor utilization of Java heap resources.

-XX:+AggressiveHeap:

Use this parameter if you are using the default throughput/parallel scavenge collector with built-in tuning enabled. The JVM can attempt to aggressively tune the parameters of its tuning algorithm based on using all the resources of the operating system on which you are running. In situations where a single product process is executing using all of the resources of the operating system, use of this option to determine if the JVM can deliver satisfactory results. Using this option while testing JVM results should reduce your tuning effort.

-XX:CMSInitiatingOccupancyFraction=75:

Configure this parameter if you are using the concurrent low-pause mark-sweep collector. This option is used to control CMS. It sets the triggering condition for when the dedicated background thread engages to conduct garbage collection on the tenured region of the heap. Unlike other garbage collection modes, the garbage collection action does not wait for an allocation to fail. Instead the objective is to trigger the garbage collection to recover sufficient space before the allocation arises that would otherwise have failed. The principle trigger is based on the percentage utilization of the Java heap, and defaults to about 70%. The default value typically ensures that CMS cycles start sufficiently, although this frequency might be higher than necessary.

However, with only a very small eden region, and no use of the survivor spaces, there is barely any opportunity for objects to age, such that the generational garbage collection support can collect short-lived objects. For systems that benefit from generational garbage collection, by producing many quite short-lived objects, the CMS defaults deny the opportunity to exploit the generational support for which the Sun HotSpot structure is primarily designed. For only a modest investment of resources in the young generation survivor spaces, and a decent eden region, to re-enable full generational garbage collection action will probably only cause an invasive pause of a second, or less, keeping the promotion of aged objects into the tenured region low. This condition gives you the full benefit of the free compaction of surviving content as objects age, and provides maximum opportunity for the CMS thread to collect the tenured region, even with large heaps.

-XX:+DisableExplicitGC:

This option disable explicit garbage collection to eliminate any unnecessary or mistimed major garbage collection cycles that might be introduced in the software components of the system.

It is recommended that developers avoid the use of System.gc() calls to cause programmer-initiated, full compaction garbage collection cycles, because such calls can interfere with tuning the resources and garbage collection for a entire application system. If you are striving to meet demanding pause time requirements, and want to prevent programmer initiated garbage collection calls, then use of this option must be strongly considered because this option causes explicit System.gc() calls to be ignored.

-XX:MaxNewSize= and -XX:NewSize=:

Use these parameters if you are using the default throughput/parallel scavenge collector, but have decided to manually tune this scavenge collector, instead of using the built-in tuning that the **-XX:+UseAdaptiveSizePolicy** parameter provides. The current young generation size is bound to be

greater than or equal to the initial or minimum young generation size, as specified on the **-XX:NewSize** parameter. This size is less than or equal to the value specified for the maximum young generation, as specified on the **-XX:MaxNewSize** parameter.

Certain circumstances might suggest that you constrain the amount of the heap that is considered by generational garbage collection, as determined by the **-XX:NewRatio** parameter, typically limiting the maximum scope of the young generation, and occasionally limiting the minimum size. For example, setting the limit of a large object that might be subject to generational garbage collection, or to limit the maximum amount of memory that is typically in use beyond the set of persistent objects with long lifetimes, you might need to set a maximum size for the young generation heap. Specifying a minimum size, for the section of the heap that is used for young generation objects, typically accompanies tuning the use of survivor spaces, which is usually of secondary importance, but must satisfy meeting the constraints for the minimum resources in the Java heap, as specified on the **-Xms** parameter.

Unless you are striving for a specific behavior within generational garbage collection, it should be unnecessary to specify either minimum or maximum separately from the use of the NewRatio option. The reasons for setting either the maximum or minimum values are typically different. Seldom do these settings need to be set to the same value, even though there is a shorthand for setting and fixing the size of the young generation section, using the **-Xmn** parameter. However, an inappropriate configuration risks the loss of the benefits of generational garbage collection entirely.

-XX:MaxPermSize (Permanent region):

Tune this parameter in conjunction with the **-Xmx** parameter to provide enough Java heap memory. The permanent region is employed to store all the class code and class-like data, such as interned strings.

The permanent region must be large enough to accommodate all of the classes that might be concurrently loaded together. Determining an appropriate size for this region might be confusing, because this region of the heap is smaller, expands more slowly, and is specifically employed for class-like objects, and it is commonly observed to be utilized at 99-100% of its current capacity. Therefore, you must be careful how you interpret out-of-memory events. You should always verify that this region is maximally expanded before providing this region with more resources.

gotcha: In any Java Platform, Enterprise Edition (Java EE) based system, with its heavy use of application class loaders, you should avoid using the **-Xnoclassgc** parameter because this parameter prevents garbage collection of this critical region of the heap, effectively creating a memory leak of class data. For a development system where you are frequently deploying changing class content, you should significantly oversize this region. You should also regularly restart this system to prevent old versions of dormant code from accumulating within currently used, and otherwise unreleaseable class loaders.

-XX:MaxTenuringThreshold=*number-of-collections****:***

Configure this parameter when using the concurrent low-pause mark-sweep collector. This parameter controls the promotion of the objects from the new generation section to the old generation section by specifying the number of collections during which an object remains in the new generation section before being moved to the old generation section. 8 is the default value.

-XX:NewRatio=2:

Use this parameter if you are using the default throughput/parallel scavenge collector, but have decided to manually tune this scavenge collector, instead of using the built-in tuning that the **-XX:+UseAdaptiveSizePolicy** parameter provides.

The Java heap is divided into two sections where objects are stored. One of the sections is where generational garbage collection occurs, and is where young generation objects reside. The other section,

which comprises the rest of the heap, is called the tenured heap, and is where the old or long-lived objects reside. This option sizes the young generation region that supports generational garbage collection, in proportion to the overall heap capacity. The current young generation size is bound to be greater-than or equal to the initial or minimum young generation size, as specified on the **-XX:NewSize** parameter. This size is less-than or equal to the value specified for the maximum young generation, as specified on the **-XX:MaxNewSize** parameter. The young generation size is maintained as a ratio to the tenured region, as determined by the current capacity of the main Java heap. The default value of 2 means that the tenured region is twice the size of the young generation region, which means that the young generation is one third of the entire Java heap.

This default value typically delivers good generational garbage collection performance, which is the typical goal of tuning the Sun HotSpot JVM. However, other strategies exist. For example, you might want to increase the proportion of the heap where generational garbage collection is conducted. If you decide to change the proportion, remember that there is a limit as to how much of the heap can reasonably be maintained by generational garbage collection, and if that limit is exceeded, all generational garbage collection might be lost because garbage collection cycles will become major garbage collection cycles over the whole heap instead of the desired minor garbage collection cycles that consider just the generational part of the heap.

2 is the default value for a server that is running in VM mode.

-XX:NewSize=128m:

Configure this parameter if you are using the concurrent low-pause mark-sweep collector. The current young generation size is bound to be greater than or equal to the initial or minimum young generation size, as specified on the **-XX:NewSize** parameter. One of the difficulties for tuning with CMS is that the worst case garbage collection times, which occurs when the CMS cycle aborts, might take several seconds, which is especially costly for a system that is employing CMS as a way to avoid long pauses.

Consequently, service level agreements might dictate the use of CMS. In this situation tuning must err on the cautious side to ensure that CMS is given every opportunity to succeed. CMS succeeds only when its anticipatory trigger ensures the CMS cycle starts early enough to always ensure that 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 called a full collection. Full collections are a sign that further tuning is required to the CMS collector to make its operation better suit your application.

-XX:+PrintTenuringDistribution:

This option is a garbage collection logging option which enables the printing of information regarding the age of the objects, tenuring information, as they age through the survivor spaces.

-XX:SurvivorRatio=:

Use this parameter if you are using the default throughput/parallel scavenge collector, but have decided to manually tune this scavenge collector, instead of using the built-in tuning that the **-XX:+UseAdaptiveSizePolicy** parameter provides, or if you are trying to tune the concurrent low pause collector.

The generational garbage collection action concerns dividing short-lived objects from longer lived ones. Only those objects that continue in use need to be preserved in the heap. The young generation region that hosts generational garbage collection includes further internal structure. It includes a large eden region where objects are initially allocated, and smaller survivor spaces where objects, that have longer lifetimes, reside. The SurvivorRatio sizes these regions in terms of how large the eden region is relative to the smaller survivor space. Sizing the survivor spaces is typically of secondary importance, because the

volume of objects that might benefit from optimization varies greatly by application. However, typically you should lower this value from the default value of 25, to something like 8.

Any changes to this parameter should be justified through an analysis of the data concerning the tenuring. You can use the **-XX:+PrintTenuringDistribution** parameter to obtain this data

gotcha: **-XX:SurvivorRatio=** *option* is incompatible with the JVM parameter **-XX:+UseAdaptiveSizePolicy**. Please use either one according to your situation.

-XX:TargetSurvivorRatio=:

Use this parameter if you are using the default throughput/parallel scavenge collector, but have decided to manually tune this scavenge collector, instead of using the built-in tuning that the **-XX:+UseAdaptiveSizePolicy** parameter provides.

This parameter encourages the JVM to increase the percentage utilization of the survivor spaces, thereby avoiding premature promotion, if possible, and maximizing the chance for objects to be collected from the young generation. The default value is 50. Better utilization of these regions might be achieved if you set this parameter to 90.

-XX:+UseAdaptiveSizePolicy:

Use this parameter to enable the built-in tuning with the default throughput/parallel scavenge collector default throughput/parallel scavenge collector.

In addition to the automatic operating system detection, Sun includes a tuning algorithm that attempts to autonomically tune the JVM to optimize the throughput goal and the efficiency of the throughput collection strategy. This tuning algorithm is turned on by default, and is explicitly engaged using the **-XX:+UseAdaptiveSizePolicy** parameter. This tuning algorithm should typically achieve satisfactory results for the majority of application workloads, saving you from having to perform additional tuning efforts. However you should still test this algorithm for your workload and verify that it meets your throughput requirements before using it in a production environment.

-XX:+UseConcMarkSweepGC:

Use this parameter to enable the concurrent low-pause mark-sweep collector. This garbage collection mode reconfigures the generational garbage collection so that your out-of-the-box the system minimizes the invasive pauses introduced by having to collect the young generation content.

This parameter also minimizes the extent of the young generation, or eden region. Server-class systems typically detect the availability of multiple processors, and attempt to collect the young generation in parallel, in an attempt to deliver absolutely the minimum pause possible, even if there exists only a limited amount of work available, giving little advantage for the use of multiple threads after the coordination overhead. To offset the work no longer being performed by generational garbage collection, it is typically necessary to increase the resources committed to the main heap by about ten to thirty percent, as compared to the throughput collector settings.

-XX:+UseParallelGC:

Use this parameter to enable the default throughput/parallel scavenge collector.

The default garbage collection mode of a server JVM is to adopt the throughput collector that conducts the minor garbage collections over the young generation in parallel as a foreground stop-the-world task. This collector can be enabled explicitly through the use of the **-XX:+UseParallelGC** parameter. Goals, other than throughput, can be specified. However, the penalty for not achieving such goals can be quite severe, and might cause a fatal out-of-memory error.

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations (distributed)

The following file paths are default locations. You can install the product and other components or create profiles in any directory where you have write access. Multiple installations of WebSphere Application Server Network Deployment products or components require multiple locations. Default values for installation actions by root and nonroot users are given. If no nonroot values are specified, then the default directory values are applicable to both root and nonroot users.

app_client_root

Table 10. Default installation root directories for the Application Client for IBM WebSphere Application Server.

This table shows the default installation root directories for the Application Client for IBM WebSphere Application Server.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppClient` (Java EE Application client only) HP-UX Linux Solaris `/opt/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\Program Files\IBM\WebSphere\AppClient`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\IBM\WebSphere\AppClient`

app_server_root

Table 11. Default installation directories for WebSphere Application Server.

This table shows the default installation directories for WebSphere Application Server Network Deployment.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppServer` HP-UX Linux Solaris `/opt/IBM/WebSphere/AppServer` Windows `C:\Program Files\IBM\WebSphere\AppServer`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppServer` Windows *user_home*`\IBM\WebSphere\AppServer`

component_root

The component installation root directory is any installation root directory described in this article. Some programs are for use across multiple components—in particular, the Web Server Plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

gskit_root

IBM Global Security Kit (GSKit) can now be installed by any user. GSKit is installed locally inside the installing product's directory structure and is no longer installed in a global location on the target system.

Table 12. Default installation directories for GSKit.

This table shows the default installation root directory for Version 8 of the GSKit, where product_root is the root directory of the product that is installing GSKit, for example IBM HTTP Server or the web server plug-in.

User	Directory
Root and nonroot	AIX HP-UX Linux Solaris *product_root*/gsk8 Windows *product_root*\gsk8

profile_root

Table 13. Default profile directories.

This table shows the default directories for a profile named profile_name on each distributed operating system.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer/profiles/*profile_name* HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer/profiles/*profile_name* Windows C:\Program Files\IBM\WebSphere\AppServer\profiles*profile_name*
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/profiles Windows *user_home*\IBM\WebSphere\AppServer\profiles

plugins_root

Table 14. Default installation root directories for the Web Server Plug-ins.

This table shows the default installation root directories for the Web Server Plug-ins for WebSphere Application Server.

User	Directory
Root	AIX /usr/IBM/WebSphere/Plugins HP-UX Linux Solaris /opt/IBM/WebSphere/Plugins Windows C:\Program Files\IBM\WebSphere\Plugins
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Plugins Windows C:\IBM\WebSphere\Plugins

wct_root

Table 15. Default installation root directories for the WebSphere Customization Toolbox.

This table shows the default installation root directories for the WebSphere Customization Toolbox.

User	Directory
Root	AIX /usr/IBM/WebSphere/Toolbox HP-UX Linux Solaris /opt/IBM/WebSphere/Toolbox Windows C:\Program Files\IBM\WebSphere\Toolbox
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Toolbox Windows C:\IBM\WebSphere\Toolbox

web_server_root

Table 16. Default installation root directories for the IBM HTTP Server.

This table shows the default installation root directories for the IBM HTTP Server.

User	Directory
Root	AIX /usr/IBM/HTTPServer HP-UX Linux Solaris /opt/IBM/HTTPServer Windows C:\Program Files\IBM\HTTPServer
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/HTTPServer Windows C:\IBM\HTTPServer

Tuning transport channel services

The transport channel services manage client connections and I/O processing for HTTP and JMS requests. These I/O services are based on the non-blocking I/O (NIO) features that are available in Java. These services provide a highly scalable foundation to WebSphere Application Server request processing. Java NIO-based architecture has limitations in terms of performance, scalability, and user usability. Therefore, integration of true asynchronous I/O is implemented. This implementation provides significant benefits in usability, reduces the complexity of I/O processing, and reduces that amount of performance tuning you must perform.

About this task

Key features of the new transport channel services include:

- Scalability, which enables the product to handle many concurrent requests
- Asynchronous request processing, which provides a many-to-one mapping of client requests to web container threads
- Resource sharing and segregation, which enables thread pools to be shared between the web container and a messaging service
- Improved usability
- Incorporation of autonomic tuning and configuration functions

Changing the default values for settings on one or more of the transport channels associated with a transport chain can improve the performance of that chain.

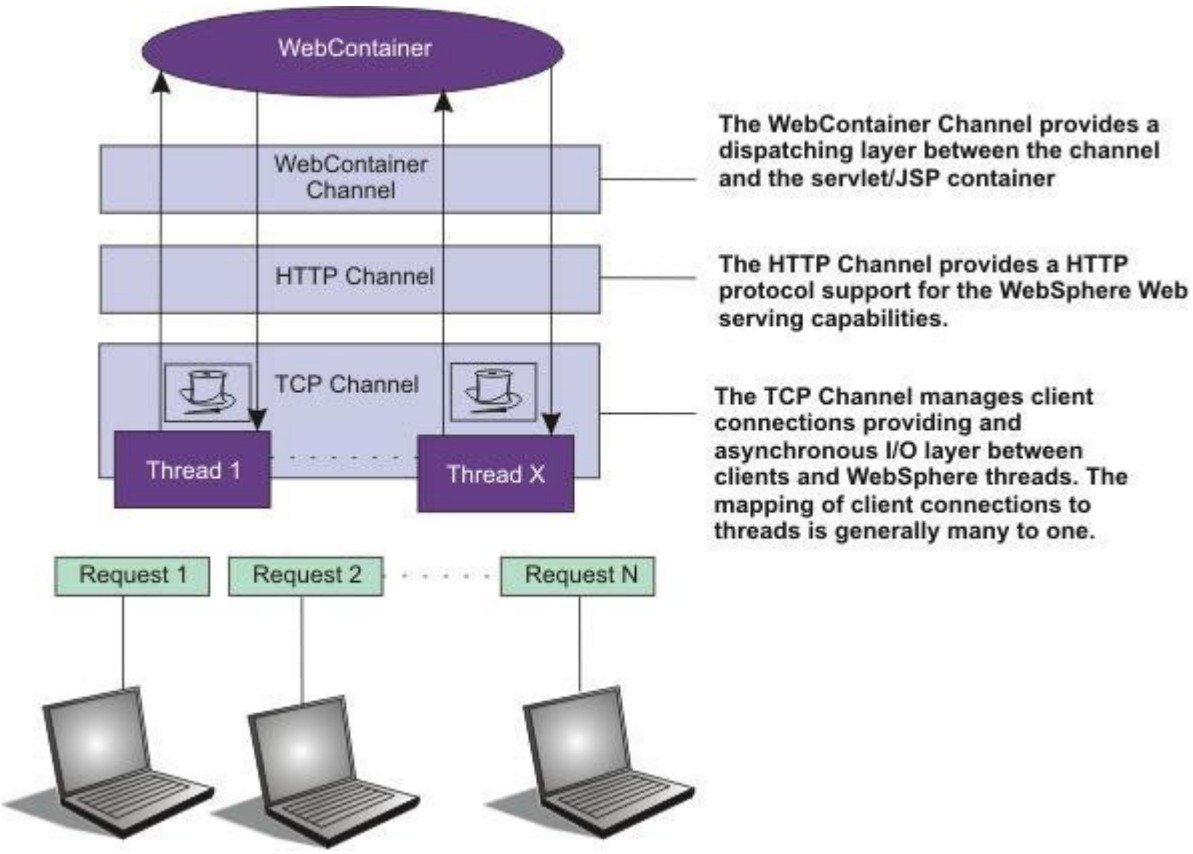


Figure 1. Transport Channel Service

Procedure

- Adjust TCP transport channel settings. In the administration console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties you are changing.
  2. Click the TCP transport channel defined for that chain.
  3. Lower the value specified for the Maximum open connections property. This parameter controls the maximum number of connections that are available for a server to use. Leaving this parameter at the default value of `20000`, which is the maximum number of connections, might lead to stalled websites under failure conditions, because the product continues to accept connections, thereby increasing the connection, and associated work, backlog. The default should be changed to a significantly lower number, such as `500`, and then additional tuning and testing should be performed to determine the optimal value that you should specify for a specific website or application deployment.
  4. If client connections are being closed without data being written back to the client, change the value specified for the Inactivity timeout parameter. This parameter controls the maximum number of connections available for a server use. After receiving a new connection, the TCP transport channel waits for enough data to arrive to dispatch the connection to the protocol-specific channels above the TCP transport channel. If not enough data is received during the time period specified for the Inactivity timeout parameter, the TCP transport channel closes the connection.

     The default value for this parameter is `60` seconds. This value is adequate for most applications. Increase the value specified for this parameter if your workload involves many connections and all of these connections cannot be serviced in 60 seconds.
  5. Assign a thread pool to a specific HTTP port. Each TCP transport channel is assigned to a particular thread pool. Thread pools can be shared between one or more TCP transport channels as well as

with other components. The default setting for a TCP transport channel is to have all HTTP-based traffic assigned to the `WebContainer` thread pool and all other traffic assigned to the `Default` thread pool. Use the **Thread pool** menu list to assign a particular thread pool to each TCP transport channel. The default setting for this parameter has all HTTP-based traffic assigned to the `WebContainer` thread pool and all other traffic is assigned to the `Default` thread pool. The information about thread pool collection describes how to create additional thread pools.

6. Tune the size of your thread pools. By default, a thread pool can have a minimum of `10` threads and a maximum of `50` maximum threads. To adjust these values, click **Thread pools** >*threadpool_name* and adjust the values specified for the Minimum Size and Maximum Size parameters for that thread pool.

   Typical applications usually do not need more than 10 threads per processor. One exception is if there is some off server condition, such as a very slow backend request, that causes a server thread to wait for the backend request to complete. In such a case, processor usage is low and increasing the workload does not increase processor throughput. Thread memory dumps show nearly all threads in a callout to the backend resource. If this condition exists, and the backend is tuned correctly, try increasing the minimum number of threads in the pool until you see improvements in throughput and thread memory dumps show threads in other areas of the run time besides the backend call.

   The setting for the Grow as needed parameter is changed unless your backend is prone to hanging for long periods of time. This condition might indicate that all of your runtime threads are blocked waiting for the backend instead of processing other work that does not involve the hung backend.

- Adjust HTTP transport channel settings. In the administration console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties you are changing.
  2. Click the HTTP transport channel defined for that chain.
  3. Tune HTTP keep-alive.

     The Use persistent (keep-alive) connections setting controls whether connections are left open between requests. Leaving the connections open can save setup and teardown costs of sockets if your workload has clients that send multiple requests. The default value is true, which is typically the optimal setting.

     If your clients only send single requests over substantially long periods of time, it is probably better to disable this option and close the connections right away rather than to have the HTTP transport channel setup the timeouts to close the connection at some later time.
  4. Change the value specified for the Maximum persistent requests parameter to increase the number of requests that can flow over a connection before it is closed.

     When the **Use persistent connections** option is enabled, the Maximum persistent requests parameter controls the number of requests that can flow over a connection before it is closed. The default value is `100`. This value should be set to a value such that most, if not all, clients always have an open connection when they make multiple requests during the same session. A proper setting for this parameter helps to eliminate unnecessary setting up and tearing down of sockets.

     For test scenarios in which the client is never closed, a value of `-1` disables the processing which limits the number of requests over a single connection. The persistent timeout shuts down some idle sockets and protects your server from running out of open sockets.
  5. Change the value specified for the Persistent timeout parameter to increase the length of time that a connection is held open before being closed due to inactivity. The Persistent timeout parameter controls the length of time that a connection is held open before being closed because there is no activity on that connection. The default value is `30` seconds This parameter is set to a value that keeps enough connections open so that most clients can obtain a connection available when they must make a request.
  6. If clients are having trouble completing a request because it takes them more than `60` seconds to send their data, change the value specified for the Read timeout parameter. Some clients pause

more than 60 seconds while sending data as part of a request. To ensure that they are able to complete their requests, change the value specified for this parameter to a length of time in seconds that is sufficient for the clients to complete the transfer of data. Be careful when changing this value that you still protect the server from clients who send incomplete data and thereby use resources (sockets) for an excessive amount of time.

7. If some of your clients require more than 60 seconds to receive data being written to them, change the value specified for the Write timeout parameter. Some clients are slow and require more than 60 seconds to receive data that is sent to them. To ensure that they are able to obtain all of their data, change the value specified for this parameter to a length of time in seconds that is sufficient for all of the data to be received. Be careful when changing this value that you still protect the server from malicious clients.

- Adjust the web container transport channel settings. In the administration console, click **Servers > Server Types > WebSphere application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties must be changed.
  2. Click the web container transport channel defined for that chain.
  3. If multiple writes are required to handle responses to the client, change the value specified for the Write buffer size parameter to a value that is more appropriate for your clients. The Write buffer size parameter controls the maximum amount of data per thread that the web container buffers before sending the request on for processing. The default value is 32768 bytes, which are sufficient for most applications. If the size of a response is greater than the size of the write buffer, the response is chunked and written back in multiple TCP writes.

     If you must change the value specified for this parameter, make sure that the new value enables most requests to be written out in a single write. To determine an appropriate value for this parameter, look at the size of the pages that are returned and add some additional bytes to account for the HTTP headers.
- **Adjust the settings for the bounded buffer.**

  Even though the default bounded buffer parameters are optimal for most of the environments, you might want to change the default values in certain situations and for some operating systems to enhance performance. Changing the bounded buffer parameters can degrade performance. Therefore, make sure that you tune the other related areas, such as the web container and ORB thread pools, before deciding to change the bounded buffer parameters.

  To change the bounded buffer parameters:
  1. In the administrative console, click **Servers > Server Types > WebSphere application servers >** *server_name*.
  2. In the Server Infrastructure section, click **Java and process management > Process definition > Java virtual machine**.
  3. Click **Custom properties**.
  4. Enter one of the following custom properties in the Name field and an appropriate value in the **Value** field, and then click **Apply** to save the custom property and its setting.
     - com.ibm.ws.util.BoundedBuffer.spins_take=*value*

       Specifies the number of times a web container thread can attempt to retrieve a request from the buffer before the thread is suspended and enqueued. This parameter enables you to trade off the cost of performing possibly unsuccessful retrieval attempts, with the cost to suspending a thread and activating it again in response to a put operation.

Information	Value
Default:	The number of processors available to the operating system minus 1.
Recommended:	Use any non-negative integer value. In practice, using an integer from 2 to 8 yields the best performance results.

Information	Value
Usage:	com.ibm.ws.util.BoundedBuffer.spins_take=6. Six attempts are made before the thread is suspended.

– com.ibm.ws.util.BoundedBuffer.yield_take=true or false

Specifies that a thread yields the processor to other threads after a set number of attempts to take a request from the buffer. Typically a lower number of attempts is preferable.

Information	Value
Default:	false
Recommended:	The effect of yield is implementation-specific for individual platforms.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_take=*boolean value*

– com.ibm.ws.util.BoundedBuffer.spins_put=*value*

Specifies the number of attempts an InboundReader thread makes to put a request into the buffer before the thread is suspended and enqueued. Use this value to trade off between the cost of repeated, possibly unsuccessful, attempts to put a request into the buffer with the cost to suspend a thread and reactivate it in response to a take operation.

Information	Value
Default:	The value of com.ibm.ws.util.BoundedBuffer.spins_take divided by 4.
Recommended:	Use any non-negative integer value. In practice an integer 2 - 8 have shown the best performance results.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_put=6. Six attempts are made before the thread is suspended.

– com.ibm.ws.util.BoundedBuffer.yield_put=true or false

Specifies that a thread yields the processor to other threads after a set number of attempts to put a request into the buffer. Typically a lower number of attempts is preferable.

Information	Value
Default:	false
Recommended:	The effect of yield is implementation-specific for individual platforms.
Usage:	com.ibm.ws.util.BoundedBuffer.yield_put=*boolean value*

– com.ibm.ws.util.BoundedBuffer.wait=*number of milliseconds*

Specifies the maximum length of time, in milliseconds, that a request might unnecessarily be delayed if the buffer is completely full or if the buffer is empty.

Information	Value
Default:	10000 milliseconds
Recommended:	A value of 10000 milliseconds usually works well. In rare instances when the buffer becomes either full or empty, a smaller value guarantee a more timely handling of requests, but there is usually a performance impact to using a smaller value.
Usage:	com.ibm.ws.util.BoundedBuffer.wait=8000. A request might unnecessarily be delayed up to 8000 milliseconds.

- Click **Apply** and then click **Save** to save these changes.

Checking hardware configuration and settings

An optimal hardware configuration enables applications to get the greatest benefit from performance tuning. The hardware speed impacts all types of applications and is critical to overall performance.

About this task

You can check hardware configuration and settings such as disk speed, system memory and processor speed to gain performance benefits.

Procedure

Use the following considerations for selecting and configuring the hardware on which the application servers run:

1. Optimize disk speed
   - Description: Disk speed and configuration have a dramatic effect on the performance of application servers running applications that are heavily dependent on the database support, using extensive messaging, or processing workflow. The disk input or output subsystems that are optimized for performance, for example Redundant Array of Independent Disks (RAID) array, high-speed drives, and dedicated caches, are essential components for optimum application server performance in these environments.

     Application servers with fewer disk requirements can benefit from a mirrored disk drive configuration that improves reliability and has good performance.
   - Recommendation: Spread the disk processing across as many disks as possible to avoid contention issues that typically occur with 1- or 2-disk systems. Placing the database tables on disks that are separate from the disks that are used for the database log files reduces disk contention and improve throughput.
2. Increase processor speed and processor cache
   - **Description:** In the absence of other bottlenecks, increasing the processor speed often helps throughput and response times. A processor with a larger L2 or L3 cache yields higher throughput, even if the processor speed is the same as a CPU with a smaller L2 or L3 cache.
3. Increase system memory
   - Description: Increase memory to prevent the system from paging memory to the disk to improve performance. Allow a minimum of 256 MB of memory for each processor and 512 MB per application server. Adjust the available memory when the system pages and the processor utilization is low because of the paging. The memory access speed might depend on the number and placement of the memory modules. Check the hardware manual to make sure that your configuration is optimal.
   - Recommendation: Use 256 MB of memory for each processor and 512 MB per application server. Some applications might require more memory.
4. Increase system memory
   - Description: Increase memory to prevent the system from paging memory to the disk to improve performance. Allow a minimum of 256 MB of memory for each processor and 512 MB per application server. Adjust the available memory when the system pages and the processor utilization is low because of the paging. The memory access speed might depend on the number and placement of the memory modules. Check the hardware manual to make sure that your configuration is optimal.
   - Recommendation: Use 256 MB of memory for each processor and 512 MB per application server. Some applications might require more memory.
5. Run network cards and network switches at full duplex

- Description: Run network cards and network switches at full duplex and use the highest supported speed. Full duplex is much faster than half duplex. Verify that the network speed of adapters, cables, switches, and other devices can accommodate the required throughput. Some websites might require multiple gigabit links.
- Recommendation Make sure that the highest speed is in use on 10/100/1000 Ethernet networks.

Tuning operating systems

You can tune your operating system to optimize the performance of WebSphere Application Server.

About this task

Tuning parameters are specific to operating systems. Because these operating systems are not WebSphere Application Server products, be aware that the products can change and results can vary.

Note: Check your operating system documentation to determine how to make the tuning parameters changes permanent and if a reboot is required.

Procedure

- Tune Windows systems
- Tune Linux systems
- Tune AIX systems
- Tune Solaris systems
- Tune HP systems

Tuning Windows systems

This topic describes how to tune Windows XP, Windows 2003, Windows 2008 and Windows Vista operating systems to optimize the performance of WebSphere Application Server. Because Windows operating systems are not WebSphere Application Server products, be aware that the products can change and results can vary.

About this task

When you have a performance concern, check the operating system settings to determine if they are appropriate for your application.

Procedure

Configure the following settings or variables according to your specific tuning needs:

- **TcpTimedWaitDelay**
  - **Description:** Determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME_WAIT state or twice the maximum segment lifetime (2MSL) state. During this time, reopening the connection to the client and server costs less than establishing a new connection. By reducing the value of this entry, TCP/IP can release closed connections faster and provide more resources for new connections. Adjust this parameter if the running application requires rapid release, the creation of new connections, or an adjustment because of a low throughput caused by multiple connections in the TIME_WAIT state.
  - **How to view or set:**
    1. Use the **regedit** command, access the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCPIP\Parameters registry subkey, and create a new REG_DWORD value named `TcpTimedWaitDelay`.
    2. Set the value to decimal `30`, which is Hex 0x0000001e. This value sets the wait time to 30 seconds.

3. Stop and restart the system.

– **Default value:** `0xF0`, which sets the wait time to 240 seconds (4 minutes).
– **Recommended value:** A minimum value of `0x1E`, which sets the wait time to 30 seconds.

- **MaxUserPort**

  gotcha: This setting is not needed for the Windows 2008 and Windows Vista operating systems. The default start port for these operating systems is 49152, and the default end port is 65535. See the Microsoft Support web site for more information.

  – **Description:** Determines the highest port number that TCP/IP can assign when an application requests an available user port from the system.
  – **How to view or set:**
    1. Use the **regedit** command, access the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TCPIP\Parameters registry subkey, and create a new REG_DWORD value named `MaxUserPort`.
    2. Set this value to at least decimal `32768`.
    3. Stop and restart the system.
  – **Default value:** None
  – **Recommended value:** At least decimal `32768`.

- **MaxConnect Backlog**
  – **Description:** If many connection attempts are received simultaneously, increase the default number of pending connections that are supported by the operating system.
  – **How to view or set:**
    1. Use the **regedit** command and access the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\AFD\Parameters registry subkey
    2. Create, if necessary, and set the following values:

       ```
       "EnableDynamicBacklog"=dword:00000001

       "MinimumDynamicBacklog"=dword:00000020

       "MaximumDynamicBacklog"=dword:00001000

       "DynamicBacklogGrowthDelta"=dword:00000010
       ```

    3. These values request a minimum of 20 and a maximum of 1000 available connections. The number of available connections is increased by 10 each time that there are fewer than the minimum number of available connections.
    4. Stop and restart the system.

- **TPC/IP acknowledgements**
  – TCP/IP can be the source of some significant remote method delays. You can increase TCP performance by immediately acknowledging incoming TCP segments, in all situations.

    To immediately acknowledge incoming TCP segments on a server that runs a Microsoft Windows XP or Windows Server 2003 operating system:
    1. Start the Registry Editor (`regedit.exe`).
    2. Locate and then click the following registry subkey:

       `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\`
    3. On the Edit menu, click **New** > **DWORD Value.**
    4. Name the new value, `TcpAckFrequency`, and assign it a value of `1`.
    5. Close the Registry Editor.
    6. Restart your Windows operating system.

- **Large page support**
  – **Description:** Using large pages can reduce the CPU overhead of managing a large JVM heap.
  – **How to view or set:** The Windows operating system provides large page support by default. Use the `-Xlp JVM` option to make use of this support.

Results

This tuning procedure improves performance of WebSphere Application Server on Windows XP, and Windows 2003 operating systems.

What to do next

After tuning your operating system for performance, consult other tuning topics for various tuning tips.

Tuning Linux systems

This topic describes how to tune the Linux operating system to optimize the performance of your WebSphere Application Server.

About this task

When you have a performance concern, check the operating system settings to determine if these settings are appropriate for your application. Because the Linux operating system is not a WebSphere Application Server product, be aware that it can change and results can vary.

Procedure

Configure the following settings and variables according to your tuning needs:

- **Changing TCP parameters**
  - **Description:** Linux offers a number of tunable TCP parameters whose default values might be sufficient for WebSphere Application Server. It might be necessary to tune these parameters in some exceptional cases. For example, you might reduce the number of sockets in specific states such as TIME_WAIT, modify the TCP keepalive operation, or modify other functions.
  - **How to view or set:**

    Consult the detailed information available under "man tcp" under your Linux distribution.
- **SUSE Linux Enterprise Server 8 (SLES 8) SP2A - sched_yield_scale tuning**
  - **Description:** The Linux scheduler is very sensitive to excessive context switching, so fixes are integrated into the SLES 8 kernel distribution to introduce delay when a thread yields processing. This fix is automatically enabled in SLES 8 SP3, but must be enabled explicitly in SLES 8 SP2A or earlier.
  - **How to view or set:**
    1. Upgrade your SLES 8 service pack to SP2A.
    2. Issue the **sysctl -w sched_yield_scale=1** command .
  - **Default value:** `0`
  - **Recommended value:** `1`
- **RedHat Advanced Server 2.1 kernel update**
  - **Description:** Kernel updates for RedHat Advanced Server 2.1 implemented changes that affect WebSphere Application Server performance, especially memory-to-memory HTTP session replication.
  - **How to view or set:**
    1. Issue the **uname -a** command
    2. If you are running any kernel prior to 2.4.9-e.23, upgrade at least to the RedHat Advanced Server 2.1 kernel, but preferably to the latest supported.
  - **Default value:** `2.4.9-e.3`
  - **Recommended value:** `2.4.9-e.23`
- **Linux file descriptors (ulimit)**
  - **Description:** Specifies the number of open files that are supported. The default setting is typically sufficient for most applications. If the value set for this parameter is too low, a file open error, memory allocation failure, or connection establishment error might be displayed.

- **How to view or set:** Check the UNIX reference pages on the **ulimit** command for the syntax of different shells. To set the **ulimit** command to `8000` for the KornShell shell (ksh), issue the **ulimit -n 8000** command. Use the **ulimit -a** command to display the current values for all limitations on system resources.
- **Default value:** For SUSE Linux Enterprise Server 9 (SLES 9), the default is `1024`.
- **Recommended value:** `8000`

- **Connection backlog**
  - **Description:** Change the following parameters when a high rate of incoming connection requests result in connection failures:

    ```
    echo 3000 > /proc/sys/net/core/netdev_max_backlog
    echo 3000 > /proc/sys/net/core/somaxconn
    ```

- **TCP_KEEPALIVE_INTERVAL**
  - **Description:** Determines the wait time between isAlive interval probes.
  - **How to view or set:** Issue the following command to set the value:

    ```
    echo 15 > /proc/sys/net/ipv4/tcp_keepalive_intvl
    ```

  - **Default value:** `75` seconds
  - **Recommended value:** `15` seconds
- **TCP_KEEPALIVE_PROBES**
  - **Description:** Determines the number of probes before timing out.
  - **How to view or set:** Issue the following command to set the value:

    ```
    echo 5  > /proc/sys/net/ipv4/tcp_keepalive_probes
    ```

  - **Default value:** `9` seconds
  - **Recommended value:** `5` seconds
- **Allocating large pages for Java virtual machine (JVM) heap (tested with SLES 9)**

  Some applications require a very large heap for optimal performance. The CPU overhead of managing a large heap can be reduced by using the "large page" support that is provided by the CPU and operating system. The following example assumes a large page size of 4MB and a desired heap size of 2300MB.

  1. Set the following three settings by a `sysctl.conf` file, typically located at `/etc/sysctl.conf`.

     Note: You must have root privilege access to modify this file. Also, verify the file is not marked as read-only before attempting to make changes.

     a. Set the number of large pages (2300MB = 575 * 4MB) by issuing the following command:

        ```
        vm.nr_hugepages = 575
        ```

     b. Set the maximum shared segment size to 2300MB plus a little more (about 95MB) (2511724800 = 2300MB * 1048576 bytes/MB + 100000000 bytes) by issuing the following command:

        ```
        kernel.shmmax = 2511724800
        ```

     c. Set the total number of memory page to be shared by issuing the following command:

        ```
        kernel.shmall = ceil(shmmax/page_size)
        ```

        This parameter sets the total number of shared memory pages that can be used system wide. Therefore, the value specified for the shmall parameter should always be at least *ceil(shmmax/page_size)* pages.

  2. Set the Xmx JVM option to `2300MB`.
  3. Relocate the program text to a lower virtual memory address (0x10000000) to provide more address space for a larger heap. On SUSE Linux Enterprise Server 9 , run the following command to relocate the text in the script that invokes the JVM or in a `.profile` file:

     ```
     echo "0x10000000" > /proc/self/mapped_base
     ```

Results

This tuning procedure improves performance of WebSphere Application Server on the Linux operating system.

What to do next

After tuning your operating system for performance, consult other tuning topics for various tuning tips.

Tuning AIX systems

This topic describes how to tune the AIX operating system to optimize the performance of your WebSphere Application Server.

About this task

There are a number of configuration changes and variables you can set to tune the performance of Websphere to suit your needs. Because the AIX operating system is not a WebSphere Application Server product, be aware that it can change and results can vary.

Procedure

Change the following configuration settings or variables according to your needs:

- **TCP_TIMEWAIT**
  - **Description:** Determines the time that must elapse before TCP/IP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME_WAIT state or twice the maximum segment lifetime (2MSL) state. During this time, reopening the connection to the client and server costs less than establishing a new connection. By reducing the value of this entry, TCP/IP can release closed connections faster, providing more resources for new connections. Adjust this parameter, if the running application requires rapid release or the creation of new connections, or if a low throughput occurs due to many connections sitting in the TIME_WAIT state.
  - **How to view or set:**

    Issue the following command to set TCP_TIMEWAIT state to 15 seconds:

    ```
    /usr/sbin/no –o tcp_timewait =15
    ```
- **AIX operating systems with DB2**
  - **Description:** Separating your DB2 log files from the physical database files can boost performance. You can also separate the log and the database files from the drive that contains the Journaled File System (JFS) service. AIX uses specific volume groups and file systems for the JFS logging.
  - **How to view or set:** Use the AIX filemon utility to view all the file system input and output and to strategically select the file system for the DB2 log files. Set the DB2 log location according to the DB2 logging information..
  - **Default value:** The default location for the DB2 log files is often the same disk drive where the database tables are stored.
  - **Recommended value:** Move the files to a disk that is separate from the DB2 data and has the minimum input or output activity.
- **AIX file descriptors (ulimit)**
  - **Description:** Specifies the various restrictions on resource usage on the user account. The **`ulimit -a`** command displays all the ulimit limits including the number of open files that are permitted. The default number of open files setting (2000) is typically sufficient for most applications. If the value set for this parameter is too low, errors might occur when opening files or establishing connections. Because this value limits the number of file descriptors that a server process might open, a value that is too low prevents optimum performance.
  - **How to view or set:** Perform the following steps to change the open file limit to 10,000 files:
    1. Open the command window.
    2. Edit the `/etc/security/limits` file. Add the following lines to the user account that the WebSphere Application Server process runs on:

       ```
       nofiles =  10000
       nofiles_hard = 10000
       ```
    3. Save the changes.

4. Restart your AIX system.
5. To verify the result, type the **ulimit -a** command on the command line.

– **Default value:** For the AIX operating system, the default setting is 2000.
– **Recommended value:** The value is application dependent and applies exclusively to application program data and the application stack.

  Increasing the ulimit file descriptor limits might improve performance. Increasing some of the other limits might be needed depending on your application. Any changes to the data or stack ulimits should ensure that data+stack < 256MB (for 32-bit WebSphere Application Server only).

  It is recommended that you change the ulimit for data to "unlimited".

- **AIX TCP_KEEPIDLE**
  – **How to view or set:**
    - If you are on an AIX operating system prior to version 5.2, use the **no** command to determine the current value or to set the value. The change is effective until the next time you restart the machine. To permanently change the value, add the **no** command to the /etc/rc.net directory. For example:

      ```
      no -o tcp_keepidle=600
      ```

    - If you are on an AIX operating system that is version 5.2 and later, use the **no -r -o** command to determine the value or to set the value. On subsequent reboots, the specified *no_optionname* value is maintained because it is written to the nextboot file. Set the **no** command as follows to enable it on future reboots:

      ```
      no -r -o arptab_size=10
      ```

  – **Default value:** 14400 half seconds (2 hours).
  – **Recommended value:** 600 half seconds (5 minutes).
- **TCP_KEEPINTVL**
  – **Description:** Specifies the interval between packets that are sent to validate the connection.
  – **How to view or set:** Use the following command to set the value to 5 seconds:

    ```
    no -o tcp_keepintvl=10
    ```

  – **Default value:** 150(1/2 seconds)
  – **Recommended value:** 10(1/2 seconds)
- **TCP_KEEPINIT**
  – **Description:** Specifies the initial timeout value for TCP connection.
  – **How to view or set:** Use the following command to set the value to 20 seconds:

    ```
    no -o tcp_keepinit=40
    ```

  – **Default value:** 150(1/2 seconds)
  – **Recommended value:** 40(1/2 seconds)
- **Allocating large pages (16 MB) for Java virtual machines heap**

  Some applications require a very large heap for optimal performance. Reduce the CPU overhead of managing a large heap by using large page support that is provided by the CPU and the operating system. The following steps allocate 4 GB of RAM as large pages (16 MB):

  1. 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
     ```

  2. After reboot, run the following command to enable large page support on the AIX operating system:

     ```
     vmo -p -o  v_pinshm=1
     ```

  3. As root user, add the following capabilities for the user:

     ```
     chuser  capabilities=CAP_BYPASS_RAC_VMM,CAP_PROPAGATE  $USER
     ```

  4. Add the -Xlp Java options to the Java command.
     a. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
     b. Under **Server Infrastructure**, click **Java and Process Management** > **Process definition** > **Java Virtual Machine**.
     c. In the **Generic JVM Argument** field, add -Xlp.

5. Add the EXTSHM custom property and set to OFF.
   a. Click **Servers** > **Server Types** > **WebSphere application servers** > ***server_name***.
   b. Under **Server Infrastructure**, click **Java and Process Management** > **Process definition** > **Environment Entries** > **New**.
   c. In the **Name** field, enter EXTSHM.
   d. In the **Value** field, enter OFF.
6. Validate large page support is used with the following command:

```
vmstat  -l  1
```

   Note: The "alp" column is non-zero when the application is running.

Enabling large pages might have serious consequences. For more details on large pages, see the information on AIX large pages.

If you do not want to use the large pages option, there is also a medium page option. The medium page size option, which is similar, and has close to the same performance gains as large pages. However, it does not involve the problems of reserving physical memory for a specific user or process. For more information, read the Tuning Java virtual machines information.

- **Other AIX information**

  Consider the other AIX operating system settings that are not within the scope of this document. You can adjust the following additional settings:
  – Adapter transmit and receive queue
  – TCP/IP socket buffer
  – IP protocol mbuf pool performance
  – Update file descriptors
  – Update the scheduler

  For more information about AIX operating systems, see the performance: resources for learning information.

Results

This tuning procedure improves performance of WebSphere Application Server on the AIX operating system.

What to do next

After tuning your operating system for performance, consult the other tuning topics for various tuning tips.

Tuning Solaris systems

The following tuning parameters are specific to the Solaris operating system. Because the Solaris operating system is not a WebSphere Application Server product, be aware that it can change and results vary.

About this task

On the Solaris operating system, WebSphere Application Server runs on the Oracle Hotspot Java virtual machine (JVM). It is important to use the correct tuning parameters with the Oracle JVM to utilize its performance optimizing features. See the JVM tuning information. Also, consider the following parameters that are specific to the Solaris operating system to ensure that WebSphere Application Server has enough resources.

Procedure

Configure the following settings or variables according to your tuning needs:

- **Solaris file descriptors (ulimit)**

- **Description:** Specifies the maximum number of open files supported. If the value of this parameter is too low, a `Too many files open` error is displayed in the WebSphere Application Server `stderr.log` file.
- **How to view or set:** Check the UNIX reference pages on the file descriptor limits for parameters and commands used. For the KornShell (ksh), the **ulimit -n** command can be used to set the desired file descriptor value and the **ulimit -a** command to display all current ulimit settings in place.
- **Default value:** 1024
- **Recommended value:** `10000`

- **Solaris TCP_TIME_WAIT_INTERVAL**
  - **Description:** Notifies TCP/IP on how long to keep the connection control blocks closed. After the applications complete the TCP/IP connection, the control blocks are kept for the specified time. When high connection rates occur, a large backlog of the TCP/IP connections accumulates and can slow server performance. The server can stall during certain peak periods. If the server stalls, the **netstat** command shows that many of the sockets that are opened to the HTTP server are in the CLOSE_WAIT or FIN_WAIT_2 state. Visible delays can occur for up to four minutes, during which time the server does not send any responses, but CPU utilization stays high, with all of the activities in system processes.
  - **How to view or set:** Use the **get** command to determine the current interval and the **set** command to specify an interval of 30 seconds. For example:

    ```
    ndd -get /dev/tcp tcp_time_wait_interval
    ndd -set /dev/tcp tcp_time_wait_interval 30000
    ```
  - **Default value:** The default time wait interval for a Solaris operating system is `240000` milliseconds, which is equal to 4 minutes.
  - **Recommended value:** `60000` milliseconds
- **Solaris TCP_FIN_WAIT_2_FLUSH_INTERVAL**
  - **Description:** Specifies the timer interval prohibiting a connection in the FIN_WAIT_2 state to remain in that state. When high connection rates occur, a large backlog of TCP/IP connections accumulates and can slow server performance. The server can stall during peak periods. If the server stalls, using the **netstat** command shows that many of the sockets opened to the HTTP server are in the CLOSE_WAIT or FIN_WAIT_2 state. Visible delays can occur for up to four minutes, during which time the server does not send any responses, but CPU utilization stays high, with all of the activity in system processes.
  - **How to view and set:** Use the **get** command to determine the current interval and the **set** command to specify an interval of 67.5 seconds. For example,

    ```
    ndd -get /dev/tcp tcp_fin_wait_2_flush_interval
    ndd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
    ```
  - **Default value:** `675000` milliseconds
  - **Recommended value:** `67500` milliseconds
- **Solaris TCP_KEEPALIVE_INTERVAL**
  - **Description:** The keepAlive packet ensures that a connection stays in an active and established state.
  - **How to view or set:** Use the **ndd** command to determine the current value or to set the value. For example:

    ```
    ndd -set /dev/tcp tcp_keepalive_interval 300000
    ```
  - **Default value:** `7200000` milliseconds
  - **Recommended value:** `15000` milliseconds
- **Solaris kernel semsys:seminfo_semopm**
  - **Description:** An entry in the `/etc/system` file can exist for this tuning parameter. This number is the maximum value of System V semaphore operations per semop call. The default value for this option is too low for highly concurrent systems.
  - **How to view or set:** Set this parameter through the `/etc/system` entry: `semsys:seminfo_semopm = 200`
  - **Default value:** None
  - **Recommended value:** 200 (100 is appropriate for most systems, but 200 might be needed in some cases.)

Note: This parameter has been superseded on the Solaris 10 operating system by the process.max-sem-ops resource control, which now has a default value of 512 per process. This default is sufficient for most applications. For more information on Solaris 10 parameters and resource controls, search for "tunable parameters" and "resource control" on the Sun Microsystems website at: http://docs.sun.com.

- **Connection backlog**
  - **Description:** Change the following parameter when a high rate of incoming connection requests result in connection failures:

    ```
    ndd -get /dev/tcp tcp_conn_req_max_q
    ndd -set /dev/tcp tcp_conn_req_max_q 8000
    ```
  - **Default value:** For Solaris 8, the default value is `128`.
  - **Default value:** For Solaris 9 and Solaris 10, the default value is `128`.
  - **Recommended value:** `8000`
- **Large page support**

  Using large pages can reduce the CPU overhead of managing a large JVM heap.

  With Solaris 9 and Solaris 10, large page support is provided by default. No operating system or JVM parameters are necessary to make use of large pages for the JVM heap.

Results

This tuning procedure improves the performance of WebSphere Application Server on the Solaris operating system.

What to do next

After tuning your operating system for performance, consult other tuning topics for various tuning tips.

Tuning HP-UX systems

This topic describes how to tune the HP-UX operating system to optimize the performance of your WebSphere Application Server. Because the HP-UX operating system is not a WebSphere Application Server product, be aware that it can change and results vary

Before you begin

On the HP-UX operating system, WebSphere Application Server runs on the Java virtual machine (JVM), which is based on the technology of Sun HotSpot JVM. Properly tuning this JVM significantly affects WebSphere Application Server performance by fully utilizing its performance optimizing characteristics. See the setting up the JVM on the HP-UX system information. It is also important to change some parameters that are specific to the HP-UX operating system to prevent WebSphere Application Server from being deprived of resources.

About this task

When you have a performance concern, check the operating system settings to determine if they are appropriate for your application.

Procedure

- Configure the following settings and variables according to your tuning needs:
  - **Tuning the HP operating system with the DB2 type 2 JDBC driver**

    When using the type 2 Java Database Connectivity (JDBC) driver on the HP operating system with DB2, you can increase the performance of WebSphere Application Server by preallocating the DB2 trace segment. Perform the following steps:

    1. Before starting application server, switch to the user that is associated with the DB2 instance.

2. Run the **db2trc alloc** command.
3. Start application server.

Usage note: Use the type 4 driver for best performance and compatibility.

Another issue with the type 2 JDBC driver on the HP operating system is code page conversion. Creating the database using the UTF-8 code set avoids this problem and significantly increases performance. See the database documentation for instructions on creating databases with a specific code set. Read the DB2 tuning parameters information.

– **The HP performance tuning parameters**

Modify HP-UX 11i settings to significantly improve WebSphere Application Server performance. For additional information about the HP performance tuning parameters, see the performance: resources for learning information.

– **Java virtual machine (JVM) virtual page size**
  - **Description:** Sets the JVM instruction and data page sizes to 64 MB to improve performance.
  - **How to view or set:** Use the **WASHOME/java/bin/SYSTEM_ARCH_PATH/java** command. The command output provides the current operating system characteristics of the process executable.
  - **Default value:** 4 MB, if not assigned
  - **Recommended value:** 64 MB

– **HP-UX 11i TCP_CONN_REQUEST_MAX**
  - **Description:** Specifies the maximum number of connection requests that the operating system can queue when the server does not have available threads. When high connection rates occur, a large backlog of TCP/IP connection requests builds up and client connections are dropped. Adjust this setting when clients start to time out after waiting to connect. Verify this situation by issuing the **netstat -p tcp** command. Look for the following value: *connect requests dropped due to full queue*
  - **How to view or set:** Set this parameter by using the **ndd -set /dev/tcp tcp_conn_request_max 8192** command.
  - **Default value:** 4096
  - **Recommended value:** In most cases the default is sufficient. Consider adjusting this value to `8192`, if the default proves inadequate.

– **HP-UX 11i kernel parameter recommendations**

Refer to the table of kernel parameters shown in the “Preparing HP-UX systems for installation” topic in the information center.

– **TCP_KEEPALIVE_INTERVAL**
  - **Description:** Determines the interval between probes.
  - **How to view or set:** Use the **ndd** command to determine the current value or to set the value. For example:

    ```
    ndd -set /dev/tcp tcp_keepalive_interval 7200000
    ```
  - **Default value:** `None`
  - **Recommended value:** `7200000` milliseconds

– **TCP_KEEPALIVES_KILL**
  - **Description:** Determines the maximum number of times to probe before dropping.
  - **How to view or set:** Use the **ndd** command to determine the current value or to set the value. For example:

    ```
    ndd -set /dev/tcp tcp_keepalives_kill 1
    ```
  - **Default value:** `1`
  - **Recommended value:** `1`

- Keeping current with the operating system and Java patches is one of the most important things you can do to optimize the performance of a server. For the latest Java patches, visit the following website:

  HP-UX Patch Information

  Also, for the latest operating system quality pack, visit the following website:

  Support Plus: Quality Pack Bundles

Results

This tuning procedure improves performance of WebSphere Application Server on the HP-UX operating system.

What to do next

After tuning your operating system for performance, consult the other tuning topics for various tuning tips.

Tuning web servers

WebSphere Application Server provides plug-ins for several web server brands and versions. Each web server operating system combination has specific tuning parameters that affect the application performance.

About this task

Following is a list of tuning parameters specific to web servers. The listed parameters may not apply to all of the supported web servers. Check your web server documentation before using any of these parameters.

Procedure

- **Tune the IBM HTTP Server 2.0.47.1, Apache 2.0.48, IBM HTTP Server 6.0, and IBM HTTP Server 6.1.** Monitoring the CPU utilization and checking the IBM HTTP Server error_log and http_plugin.log files can help you diagnose web server performance problems.

  You can also configure the IBM HTTP Server to show a status page:
  - Edit the IBM HTTP Server httpd.conf file and remove the comment character (#) from the following lines in this file:

    ```
    #LoadModule status_module, modules/ApacheModuleStatus.dll,
    #<Location/server-status>
    #SetHandler server-status
    #</Location>
    ```
  - Save the changes and restart the IBM HTTP Server.
  - In a web browser, go to: http://*your_host*/server-status. Alternatively, click **Reload** to update status.
  - (Optional) If the browser supports refresh, go to http://*your_host*/server-status?refresh=5 to refresh every five seconds.

  All of these web servers allocate a thread to handle each client connection. Ensuring that enough threads are available for the maximum number of concurrent client connections helps prevent this tier from being a bottleneck. The settings for these web servers can be tuned by making changes to the httpd.conf file on the web server system.

  You can check the IBM HTTP Server error_log file to see if there are any warnings about having reached the maximum number of clients (MaxClients). There are several parameters, depending on the specific operating system platform, that determine the maximum number of clients the web server supports. See http://httpd.apache.org/docs-2.0/mod/mpm_common.html#maxclients for a description of the MaxClients parameters.
- **Support thousands of concurrent clients.** It is not unusual for a single IBM HTTP Server system to support thousands of concurrent clients. If your requirements are to support more concurrent clients than the number of threads that are supported by the web server operating system and hardware, consider using multiple web servers.
- **Respond to a Connection Refused error message**. Some clients might receive a Connection Refused error message if there is a sudden increase in the number of clients. Increasing the ListenBacklog and StartServer parameters can reduce or eliminate this error.
  - The ListenBacklog parameter indicates to the operating system the maximum allowed number of pending connections. Although the IBM HTTP Server default is 511, the actual value can be much

higher or lower depending on the corresponding operating system parameter. To handle large numbers of simultaneous connections, this parameter and the corresponding OS parameter might need to be set to the number (possibly thousands) of expected simultaneous connections. (See the information about tuning operating systems for additional information on how to tune your operating system.

  - The StartServers parameter indicates the number of IBM HTTP Server processes to initially start. Pre-starting these IBM HTTP Server threads/processes reduces the chance of a user having to wait for a new process to start. You should set this parameter to a value equal to the MinSpareServers parameter so that the minimum number of IBM HTTP Server processes needed for this client load is started immediately.

- **Prevent the frequent creation and destruction of client threads/processes as the number of users change.** You can a use the MinSpareServers and MaxSpareServers to specify the minimum and maximum number of servers (client threads/processes) that can exist in an idle state. To prevent frequent creation and destruction of client threads/processes as the number of users change, set this range large enough to include the maximum number of simultaneous users.
- **Change the setting on the web server's Access logging parameter to reduce the load on the web server.** If you do not need to log every access to the Application Server, change the default value of the web server's Access logging parameter. This change will reduce the load on the web server.
- **Modify the settings of the Load balancing option and Retry interval web server plug-in properties to improve performance.** You can improve the performance of IBM HTTP Server (with the WebSphere web server plug-in) by modifying the following web server plug-in configuration properties:
  - Load balancing option, which specifies the load balancing option that the plug-in uses in sending requests to the various application servers associated with that web server.

    The goal of the default load balance option, Round Robin, is to provide an even distribution of work across cluster members. Round Robin works best with web servers that have a single process sending requests to the Application Server. If the web server is using multiple processes to send requests to the Application Server, the Random option can sometimes yield a more even distribution of work across the cluster.
  - Retry interval value, which specifies the length of time to wait before trying to connect to a server that has been marked temporarily unavailable.

    How can lowering the retry interval affect throughput ? If the plug-in attempts to connect to a particular application server and that application server is offline or in the process of restarting, the requests must wait for a timeout period. This process causes delayed responses for those requests. If you set the retry interval value too high, then an available application server is not utilized.

    Specify the retry interval value based on the following factors:

    - How long it will take for your application servers to restart
    - How averse you are to the delay caused by retrying too often
    - How important it is to utilize all of your application servers

  Making these changes can help the IBM HTTP Server to support more product users. To modify these properties, in the administrative console, click **Servers > Server Types > Web Servers >** *web_server_name* **> Plug-in properties > Request routing** .

Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories imply specific default directory locations. This article describes the conventions in use for WebSphere Application Server.

Default product locations (distributed)

The following file paths are default locations. You can install the product and other components or create profiles in any directory where you have write access. Multiple installations of WebSphere Application Server Network Deployment products or components require multiple locations. Default values for

installation actions by root and nonroot users are given. If no nonroot values are specified, then the default directory values are applicable to both root and nonroot users.

app_client_root

Table 17. Default installation root directories for the Application Client for IBM WebSphere Application Server.

This table shows the default installation root directories for the Application Client for IBM WebSphere Application Server.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppClient` (Java EE Application client only) HP-UX Linux Solaris `/opt/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\Program Files\IBM\WebSphere\AppClient`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppClient` (Java EE Application client only) Windows `C:\IBM\WebSphere\AppClient`

app_server_root

Table 18. Default installation directories for WebSphere Application Server.

This table shows the default installation directories for WebSphere Application Server Network Deployment.

User	Directory
Root	AIX `/usr/IBM/WebSphere/AppServer` HP-UX Linux Solaris `/opt/IBM/WebSphere/AppServer` Windows `C:\Program Files\IBM\WebSphere\AppServer`
Nonroot	AIX HP-UX Linux Solaris *user_home*`/IBM/WebSphere/AppServer` Windows *user_home*`\IBM\WebSphere\AppServer`

component_root

The component installation root directory is any installation root directory described in this article. Some programs are for use across multiple components—in particular, the Web Server Plug-ins, the Application Client, and the IBM HTTP Server. All of these components are part of the product package.

gskit_root

IBM Global Security Kit (GSKit) can now be installed by any user. GSKit is installed locally inside the installing product's directory structure and is no longer installed in a global location on the target system.

Table 19. Default installation directories for GSKit.

This table shows the default installation root directory for Version 8 of the GSKit, where product_root is the root directory of the product that is installing GSKit, for example IBM HTTP Server or the web server plug-in.

User	Directory
Root and nonroot	AIX HP-UX Linux Solaris *product_root*`/gsk8` Windows *product_root*`\gsk8`

profile_root

Table 20. Default profile directories.

This table shows the default directories for a profile named profile_name on each distributed operating system.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer/profiles/*profile_name* HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer/profiles/*profile_name* Windows C:\Program Files\IBM\WebSphere\AppServer\profiles*profile_name*
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/profiles Windows *user_home*\IBM\WebSphere\AppServer\profiles

plugins_root

Table 21. Default installation root directories for the Web Server Plug-ins.

This table shows the default installation root directories for the Web Server Plug-ins for WebSphere Application Server.

User	Directory
Root	AIX /usr/IBM/WebSphere/Plugins HP-UX Linux Solaris /opt/IBM/WebSphere/Plugins Windows C:\Program Files\IBM\WebSphere\Plugins
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Plugins Windows C:\IBM\WebSphere\Plugins

wct_root

Table 22. Default installation root directories for the WebSphere Customization Toolbox.

This table shows the default installation root directories for the WebSphere Customization Toolbox.

User	Directory
Root	AIX /usr/IBM/WebSphere/Toolbox HP-UX Linux Solaris /opt/IBM/WebSphere/Toolbox Windows C:\Program Files\IBM\WebSphere\Toolbox
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Toolbox Windows C:\IBM\WebSphere\Toolbox

web_server_root

Table 23. Default installation root directories for the IBM HTTP Server.

This table shows the default installation root directories for the IBM HTTP Server.

User	Directory
Root	AIX /usr/IBM/HTTPServer HP-UX Linux Solaris /opt/IBM/HTTPServer Windows C:\Program Files\IBM\HTTPServer

Table 23. Default installation root directories for the IBM HTTP Server (continued).

This table shows the default installation root directories for the IBM HTTP Server.

User	Directory
Nonroot	AIX HP-UX Linux Solaris *user_home*/IBM/HTTPServer Windows C:\IBM\HTTPServer

Using PassByReference optimization in SCA applications

Support exists for the @AllowsPassByReference annotation, which can be used to bypass marshaling and unmarshaling when a client invokes a service located in the same JVM over a remote interface.

About this task

Typically, a performance intensive aspect of service invocations is data marshaling and unmarshaling. Though invocation over a local interface always results in pass-by-reference semantics so that no data is copied, an invocation over a remotable interface entails pass-by-value semantics, which typically results in copying of the data which can be expensive.

The SCA default binding provides the @AllowsPassByReference as an optimization that you can use on your service implementation at the class level or at the individual method level.

In placing the @AllowsPassByReference annotation on the service implementation class or methods, the implementor agrees not to modify the data in a way that would violate the pass-by-value semantics. This allows both client and service to assume they are working with their own copy of the data even though the runtime environment has optimized to not perform the actual data serialization and deserialization, to save this expense.

Parameters, return types, and business exceptions are passed by reference if the service implementation class has the @AllowsPassByReference annotation defined at the class level or individual method level.

More specifically, the PassByReference optimization is performed when all of the following are true:

- Client and service have been targeted to the same JVM.
- The invocation is over the default binding.
- @AllowsPassByReference is present. Either the service implementation is a Java implementation with an appropriate @AllowsPassByReference annotation, or a composite implementation, ultimately recursively implemented in terms of such an @AllowsPassByReference-annotated Java implementation.
- All input, output, and exception types have the same package-qualified class names and can be loaded by a class loader common to or shared by both client and service.
- Both client and service are part of the same business-level application.

  This requirement applies to OSOA SCA applications. For OSOA SCA applications, both client and service must be part of the same business-level application. This requirement does not apply to OASIS SCA applications.

Procedure

1. To enable PassByReference optimization for SCA applications, ensure all classes that you want to optimize are loaded by the same class loader. Use the SCA contribution import and export support.
2. Create a Java archive (JAR) file that contains all classes that are loaded by the same class loader during both client and service execution.
3. Add an `sca-contribution.xml` file to the `META-INF` directory in the JAR.

See the OSOA Assembly specification for information on `sca-contribution.xml`. The contribution definition must contain an `export.java` statement that exports all packages contained in the JAR that are accessed by either the client or service JAR file. For example:

```
<contribution xmlns="http://www.osoa.org/xmlns/sca/1.0"
    targetNamespace="http://test.sca.scafp/pbr/shared/java">
    <export.java package="com.ibm.sample.interface"/>
    <export.java package="com.ibm.sample.types"/>
</contribution>
```

If the client and service JAR files are not already using an `sca-contribution.xml` file, update these files to use a contribution definition that imports the packages that are exported by the shared library. For example, the contribution files for the client and service that access the previous shared contribution might look like this:

```
Client:
<contribution xmlns="http://www.osoa.org/xmlns/sca/1.0"
    targetNamespace="http://test.sca.scafp/pbr/shared"
    xmlns:pbr="http://test.sca.scafp/pbr/shared">
    <deployable composite="pbr:PassByRef.SharedClient"/>
    <import.java package="com.ibm.sample.interface"/>
    <import.java package="com.ibm.sample.types"/>
</contribution>

Service:
<contribution xmlns="http://www.osoa.org/xmlns/sca/1.0"
    targetNamespace="http://test.sca.scafp/pbr/shared"
    xmlns:pbrsh="http://test.sca.scafp/pbr/shared">
    <deployable composite="pbrsh:PassByRef.SharedService"/>
    <import.java package="com.ibm.sample.interface"/>
    <import.java package="com.ibm.sample.types"/>
</contribution>
```

4. Deploy the SCA application

   Add the client, service, and shared contributions as an asset into the WebSphere repository. This can occur in any order, however you must add the shared contribution as an asset before you can add either the client or service asset as a composition unit to a business-level application. Only the client and service assets need to be added as composition units to your business-level applications. During the add composition unit operation for both the client and the service, the shared asset is automatically added to the business-level application as a shared library.

   Clients that are deployed outside of a business-level application cannot use the PassByReference optimization to invoke SCA services deployed inside a business-level application. For example, a user-created web application archive (WAR) file using the default binding cannot be installed into a business-level application, and therefore a WAR-hosted client might not participate in the PassByReference optimization. The PassByReference optimization is supported only between JAR files.

   For OSOA SCA applications, you must install both service JAR files in the same business-level application.

   For OASIS SCA applications, the JAR files can be in separate business-level applications.

Results

You have enabled PassByReference optimization for SCA applications.

Tuning the application server using pre-defined tuning templates

You can use the python-based tuning script, applyPerfTuning.py, along with one of its template files, to apply pre-defined performance tuning templates to your application server or cluster. The script, and these property-based template files are located in the *WAS_HOME*/bin directory.

Before you begin

bprac: The configuration settings applied by this script and the associated tuning templates should be viewed as potential performance tuning options for you to explore or use as a starting point for additional tuning. The configuration settings that each of the pre-defined templates applies are geared towards optimizing common application server environments or scenarios. Typically, these settings improve performance for many applications.

Because optimizing for performance often involves trade-offs with features, capabilities, or functional behavior, some of these settings might impact application correctness, while other settings might be inappropriate for your environment. Please review the documentation below and consider the impact of these settings to your application inventory and infrastructure.

As with any performance tuning exercise, the settings configured by the predefined templates should be evaluated in a controlled preproduction test environment. You can then create a customized template to refine the tuning settings to meet the specific needs of your applications and production environment.

Note: This topic references one or more of the application server log files. As a recommended alternative, you can configure the server to use the High Performance Extensible Logging (HPEL) log and trace infrastructure instead of using `SystemOut.log` , `SystemErr.log`, `trace.log`, and `activity.log` files on distributed and IBM i systems. You can also use HPEL in conjunction with your native z/OS logging facilities. If you are using HPEL, you can access all of your log and trace information using the LogViewer command-line tool from your server profile bin directory. See the information about using HPEL to troubleshoot applications for more information on using HPEL.

Typically, when you run the applyPerfTuning.py script, you will specify either the production.props template file or the development.props template file to apply against the target server or cluster.

- If you specify the production.props template file when you run the applyPerfTuning.py script, the script applies configuration settings that are appropriate for a production environment where application changes are rare and optimal runtime performance is important.
- If you specify the development.props template file when you run the applyPerfTuning.py script, the script applies configuration settings that are appropriate for a development environment where frequent application updates are performed and system resources are at a minimum.

In addition to these two common templates, a third template file, default.props, is provided to enable you to revert the server configuration settings back to the out-of-the-box defaults settings.

You can also create your own custom tuning template. To create a custom tuning template, copy one of the existing templates, modify the configuration settings to better fit the needs of your applications and environment, and then use the applyPerfTuning.py script to apply these customized settings. The script and properties files leverage the property file configuration management features that wsadmin provides, and can easily be augmented to tune additional server components. See the topic Using properties files to manage system configuration for more information.

About this task

Review the following table to see the configuration changes that occur based on the template file that you specify when you run the applyPerfTuning.py script. A blank cell in this table indicates that the listed parameter is not configured, or is configured back to the default settings for the server defaults.

Table 24. Tuning parameters and their template values. The table includes the tuning parameter and its value for the default template, the production template and the development template.

Parameter	Server default (default.props template file)	Production environment (production.props template file)	Development environment (development.props template file)
JVM Heap Size (MB) See the topic Tuning the IBM virtual machine for Java for more information about this setting.	50 min / 256 max	512 min / 512 max	256 min / 512 max
Verbose GC See the topic Tuning the IBM virtual machine for Java for more information about this setting.	disabled	enabled	disabled
JVM Diagnostic Trace (Generic JVM Arguments) See the topic Tuning the IBM virtual machine for Java for more information about this setting. **gotcha:** This setting might cause issues when web services are used in certain scenarios. Therefore, if you are running web services, and are experiencing throughput optimization issues, you can remove this parameter from the script, or set the opti level to `0`.	-Dcom.ibm.xml.xlxp.jaxb .opti.level=3	-Dcom.ibm.xml.xlxp.jaxb .opti.level=3	-Dcom.ibm.xml.xlxp.jaxb .opti.level=3
HTTP (9080) and HTTPS (9443) Channel maxKeepAliveRequests See the topic HTTP transport custom properties for more information about this setting.	100	10000	10000
TCP Channel maxOpenConnections	20000	500	500
TCP Channel listenBacklog	511	128	128
Development Mode See the topic Application server settings for more information about this setting.	disabled		enabled

Table 24. Tuning parameters and their template values (continued). The table includes the tuning parameter and its value for the default template, the production template and the development template.

Parameter	Server default (default.props template file)	Production environment (production.props template file)	Development environment (development.props template file)
Server Component Provisioning See the topic Application server settings for more information about this setting.	disabled	enabled	enabled
PMI Statistic Set See the topic Enabling PMI data collection for more information about this setting.	basic	none	none
Authentication Cache Timeout See the topic Authentication cache settings for more information about this setting.	10 minutes	60 minutes	60 minutes
Data Source Connection Pool Size* See the topic Connection pool settings for more information about this setting.	1 min / 10 max	10 min / 50 max	
Data Source Prepared Statement Cache Size* See the topic WebSphere Application Server data source properties for more information about this setting.	10	50	
ORB Pass-by-Reference** See the topic Request Broker service settings for more information about this setting.	disabled	enabled	enabled
Web Server Plug-in ServerIOTimeout	900	900	900
Thread Pools (Web Container, ORB, Default) See the topic Thread pool settings for more information about this setting.	50 min / 50 max, 10 min / 50 max, 20 min / 20 max		5 min / 10 max

Table 24. Tuning parameters and their template values (continued). The table includes the tuning parameter and its value for the default template, the production template and the development template.

Parameter	Server default (default.props template file)	Production environment (production.props template file)	Development environment (development.props template file)
Table notes: * Indicates items that are tuned only if they exist in the configuration. For example, a data source connection pool typically does not exist until an application is installed on the application server. If these items are created after your run the script, they are given the standard server default values unless you specify other settings. ** Enabling ORB Pass-By-Reference can cause incorrect application behavior in some cases, because the Java EE standard assumes pass-by-value semantics. However, enabling this option can improve performance up to 50% or more if the EJB client and server are installed in the same instance, and your application is written to take advantage of these feature. The topic Object Request Broker service settings can help you determine if this setting is appropriate for your environment.			

Following are a few subtle platform-specific tuning differences:

Solaris **Solaris platform**

The following Generic JVM arguments are used for both the production and development environments:

-XX:-UseAdaptiveSizePolicy

-XX:+UseParallelGC

-XX:+AggressiveOpts

-XX:+UnlockDiagnosticVMOptions -server

-Dcom.ibm.xml.xlxp.jaxb.opti.level=3

HP-UX **HP-UX platform**

The following Generic JVM arguments are used for both the production and development environments:

-XX:+AggressiveOpts

-XX:+ForceMmapReserved

-XX:SurvivorRatio=16

-XX:+UseParallelGC

-Djava.nio.channels.spi.SelectorProvider=sun.nio.ch.DevPollSelectorProvider

-XX:-ExtraPollBeforeRead -XX:+UseSpinning

-Dcom.ibm.xml.xlxp.jaxb.opti.level=3

Procedure

- Start the wsadmin tool if it is not already running, and then complete one of the following actions to tune an application server or all of the application servers in a cluster.
- Run the applyPerfTuning.py script to tune a specific server or cluster of servers running in a production environment.

```
wsadmin -f applyPerfTuningTemplate.py
[-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile production.props
```

- Run the applyPerfTuning.py script to tune a specific server or cluster of servers running in a development environment.

```
wsadmin -f applyPerfTuningTemplate.py
[-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile development.props
```

- Run the applyPerfTuning.py script to change the settings for a server or a cluster back to the standard out-of-the-box default configuration settings.

```
wsadmin -f applyPerfTuningTemplate.py
[-nodeName node_name -serverName server_name][clusterName cluster_name] -templateFile default.props
```

What to do next

Conduct a performance evaluation, and tuning exercise to determine if you should further fine tune the server for your specific applications.

Chapter 6. Troubleshooting performance problems

This topic illustrates that solving a performance problem is an iterative process and shows how to troubleshoot performance problems.

Before you begin

It is recommended that you review the tuning parameters hot list page before reading this topic.

About this task

Solving a performance problem is frequently an iterative process of:

- Measuring system performance and collecting performance data
- Locating a bottleneck
- Eliminating a bottleneck

This process is often iterative because when one bottleneck is removed the performance is now constrained by some other part of the system. For example, replacing slow hard disks with faster ones might shift the bottleneck to the CPU of a system.

Measuring system performance and collecting performance data

- 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.
- If the performance problem under investigation only occurs when a large number of clients use the system, then the benchmark must also simulate multiple users. 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. The performance data that is provided by WebSphere Application Server Performance Monitoring Infrastructure (PMI) helps to monitor and tune the application server performance. See the information on why use request metrics to learn more about performance data that is provided by WebSphere Application Server. Request metrics allows a request to be timed at WebSphere Application Server component boundaries, enabling a determination of the time that is spent in each major component.

Locating a bottleneck

Consult the following scenarios and suggested solutions:

- **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 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. See the *Administering applications and their environment* PDF for more information.

- **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.

Eliminating a bottleneck

Consider the following methods to eliminate a bottleneck:

- 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.

See the *Administering applications and their environment* PDF for more information.

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.

Notices

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.

APACHE INFORMATION. This information may include all or portions of information which IBM obtained under the terms and conditions of the Apache License Version 2.0, January 2004. The information may also consist of voluntary contributions made by many individuals to the Apache Software Foundation. For more information on the Apache Software Foundation, please see http://www.apache.org. You may obtain a copy of the Apache License at http://www.apache.org/licenses/LICENSE-2.0.

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 Intellectual Property & Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

Trademarks and service marks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. For a current list of IBM trademarks, visit the IBM Copyright and trademark information Web site (www.ibm.com/legal/copytrade.shtml).

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.

Index