WebSphere® Application Server for Distributed Platforms, Version 7.0

Tuning guide

Note

Before using this information, be sure to read the general information under “Notices” on page 115.

Compilation date: September 9, 2008

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 e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-5250.

  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.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Changes to serve you more quickly

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.

Under construction!

The Information Development Team for IBM WebSphere Application Server is changing its PDF book delivery strategy to respond better to user needs. The intention is to deliver the content to you in PDF format more frequently. During a temporary transition phase, you might experience broken links. During the transition phase, expect the following link behavior:

- 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

Thanks for your patience, in the short term, to facilitate the transition to more frequent PDF book updates.

Chapter 1. Planning for performance

How well a Web site 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.

- 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.
- See the documentation for the product functionality to improve performance .

Application design consideration

This topic describes the architectural suggestions in design and how to tune applications.

Consult the Designing applications topic in the *Developing and deploying applications* PDF, which highlights Web sites 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.

The Designing applications topic in the *Developing and deploying applications* PDF contains the architectural suggestions in design 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.

Note: Use the following information as an architectural guide when implementing applications:

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

Persistence

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.

Model-view-controller pattern

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.

Statelessness

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 2.1 container tuning
- “Tuning dynamic cache with the cache monitor” on page 104

Caching

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.

Asynchronous considerations

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 e-mail 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.

Third-party libraries

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 reference:

IBM WebSphere Developer Technical Journal: The top 10 (more or less) Java EE best practices

Improve performance in your XML applications, Part 2

Chapter 2. 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.

- Use this product functionality to improve performance.

  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.

  Details are available in the following topics:

  – “Application design consideration” on page 1 (architectural suggestions)

  – Designing applications.

    See the *Developing and deploying applications* PDF for more information.(coding best practices)

Chapter 3. 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 Web site. 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.

- 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 (TPV) 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 TPV.

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

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

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.

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 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 “Using the Performance and Diagnostic Advisor” on page 11 for startup and configuration steps.

Diagnostic alerts

In WebSphere Application Server Version 7.0 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. See the *Administering applications and their environment* PDF for more information. 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 below 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 topics in the *Administering applications and their environment* PDF, 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 below 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 topic in the *Administering applications and their environment* PDF. 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 in the *Administering applications and their environment* PDF 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 documentation in the *Administering applications and their environment* PDF.

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 . 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.

1. Ensure that PMI is enabled, which is default. If PMI is disabled, consult the Enabling PMI using the administrative console topic. 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. Click **Servers** > **Application servers** in the administrative console navigation tree.
3. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. Click **Apply**.
10. Click **Save**.
11. Click the **Runtime** tab.
12. 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.
13. 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 topic, Performance: Resources for learning in the subsection of Monitoring performance with third-party tools.
14. 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.
15. Click **OK**.
16. 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.
17. 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 topic, “Advice configuration settings” on page 14.

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** > **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** > **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.

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.
```

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:

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.

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 15 topic.

Lightweight memory leak detection

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

Memory leaks in Java applications

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.

Lightweight memory leak detection in WebSphere Application Server

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:

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 15 topic. If a memory leak is detected and you want to find the heap dump, refer to the Locating and analyzing heap dumps topic.

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. On a Java virtual machines (JVM) in WebSphere Application Server, you cannot enable automated heap dump generation. You might 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.

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server runtime operations. WebSphere Application Server supports the Jacl and Jython scripting languages only. To learn more about the wsadmin tool, see the *Administering applications and their environment* PDF for more information.

1. Start the wsadmin scripting client. You have several options to run scripting commands, ranging from running them interactively to running them in a profile.
2. Invoke the generateHeapDump operation on a JVM MBean, for example,
   - Finding JVM objectName:

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

   - Invoking the generateHeapDump operation on JVM MBean:

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

     where,

`$`	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

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.

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.

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 Web site.

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 (TPV) 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 TPV. The performance advisor in TPV provides more extensive advice than the “Performance and Diagnostic Advisor” on page 9. For example, TPV provides advice on setting the dynamic cache size, setting the Java virtual machine (JVM) heap size and using the DB2 Performance Configuration wizard.

1. Enable PMI in the application server as described in the Enabling PMI using the administrative console article.

   To monitor performance data through the PMI interfaces, you must first enable PMI through the administrative console before restarting the server.
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 TPV uses the extended monitoring level; however, the performance advisor in TPV 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. Performance Monitoring Infrastructure 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 article, Performance: Resources for learning in the subsection of Monitoring performance with third party tools.
5. Log performance data with TPV.
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 TPV 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 TPV logs, the logs must be at least 300 seconds of duration.

   For more information about configuring user and logging settings of TPV, refer to the Configuring TPV settings article.
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 (TPV) by clicking the Advisor link in TPV for a server.

For more information on how to use the performance advisor in TPV, 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 4. 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 Web site 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 link above 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.

Chapter 5. 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 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 exmaple, 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. To change the system wide value, perform the following steps:

Results

Repeat this process until you determine the ideal buffer size.

What to do next

The TCP/IP buffer sizes are changed. Repeat this process until you determine the ideal buffer size.

Chapter 6. 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

- 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 a Sun HotSpot JVM, see the topic *Tuning Sun HotSpot Java virtual machines (Solaris & HP-UX)*.
- 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

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.

1. Limit the number of dumps that are taken in specific situations.

   In certain error conditions, multiple application server threads might fail and the JVM requests a TDUMP for each of those threads. If a significant number of threads fail at the same time, the resulting number of TDUMPs that are taken concurrently might lead to other system problems, such. as a shortage of auxiliary storage. Use the JAVA_DUMP_OPTS environment variable to specify the number of dumps that you want the JVM to produce in certain situations. The value specified for this variable does not affect the number of TDUMPS that are generated because of com.ibm.jvm.Dump.SystemDump() calls from applications that are running on the application server.

   For example, if you want to configure JVM such that it:

- Limits the number of TDUMPs that are taken to one
- Limits the number of JAVADUMPs taken to a maximum of three
- Does not capture any documentation if an INTERRUPT occurs

Then, set the JAVA_DUMP_OPTS variable to the following value:

```
JAVA_DUMP_OPTS=ONANYSIGNAL(JAVADUMP[3],SYSDUMP[1]),ONINTERRUPT(NONE)
```

2. 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.

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

3. 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.

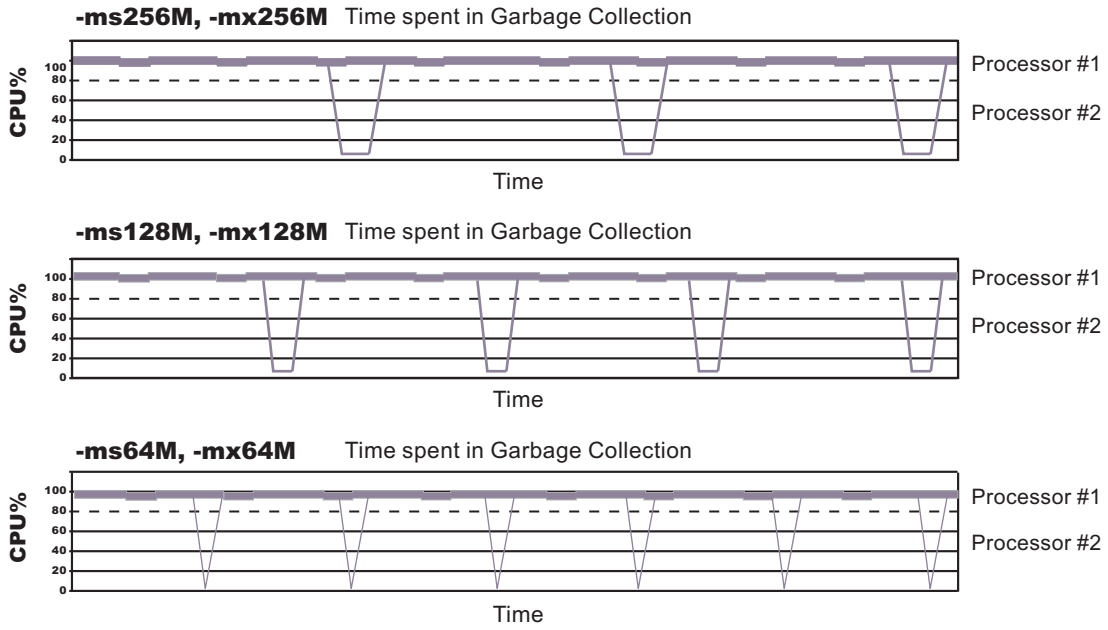
For performance analysis, the initial and maximum heap sizes should be equal.

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.

Varying Java Heap Settings



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.

for the controller and the servant if the server is configured to run in 64-bit mode.

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.

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.

Default	256 MB
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.

- 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.

- 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 instead of 4 KB pages, which 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 enable the use of 64KB pages.

AIX 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.

Default	4 KB
Recommended	-Xlp64k enables the 64 KB page size support. 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.

4. 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. These application crashes if the applications are having memory leaks because the high workload accelerates the magnification of the leakage, and a memory allocation failures 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.

5. 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® Web site.

   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.

– optthruput is the default policy, and 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 optthruput 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.
– gencon, is the policy that 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.
– 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 p™ System p and System z™ processors. The subpool policy is similar to the optthruput policy except that the heap is divided into subpools that provide improved scalability for object allocation.

Default	optthruput
Recommended	optthruput
Usage	Specifying `Xgcpolicy:optthruput` sets the garbage collection policy to optthruput

Setting **gcpolicy** to `optthruput` disables concurrent mark. You should get optimal throughput results when you use the optthruput 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. Therefore, class unloading can decrease performance.

  You can use the `-Xnoclassgc` argument to disable class garbage collection so that your applications can reuse classes more easily. Turning off class garbage collection eliminates the overhead of loading and unloading the same class multiple times.

  Note: This argument should be used with caution, if your application creates classes dynamically, or uses reflection, because for this type of application, the use of this option can lead to native memory exhaustion, and cause the JVM to throw an Out-of-Memory Exception. When this option is used, if 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.

Default	Class garbage collection is enabled.
Recommended	Disable class garbage collection.
Usage	Specify `Xnoclassgc` to disable class garbage collection.

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.

Note: 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.

Default	The Share classes in a cache option are enabled.
Recommended	Leave the share classes in a cache option enabled.
Usage	Specifying `-Xshareclasses:none` disables the share classes in a cache option.

7. Enable compressed references on 64-bit environments, such as AIX 64, Linux PPC 64, zLinux 64, and Microsoft Windows AMD64, and 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.

   Note:

   The compressed references feature is not supported on:

   - Solaris 64-bit JVM
   - 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 have to determine whether configuration update performance or consistency checking is more important. When configuration consistency checking is turned on, a significant amount of time might be required to save a configuration change, or to deploy a several 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 **Servers > Server Types > WebSphere application servers >** *server_name*.
   b. In the Server Infrastructure section, click **Java and process management > Process definition**.
   c. In the Additional Properties section , click **Java virtual machine > Custom properties > New**.
   d. Enter `config_consistency_check` in the **Name** field and `false` in the **Value** field.
   e. Click **APPLY**.
   f. Click **Save** to save your changes to the master configuration.
   g. Restart the server.

   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.

Chapter 7. 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

- 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.

- 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 these tuning parameters:

  - -XX:+DisableExplicitGC.

- 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.

Note: 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*

Note:

- 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 Web site, 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.

Note: 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

Note: **-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.

Chapter 8. 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 end 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 have to 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 and
- 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.

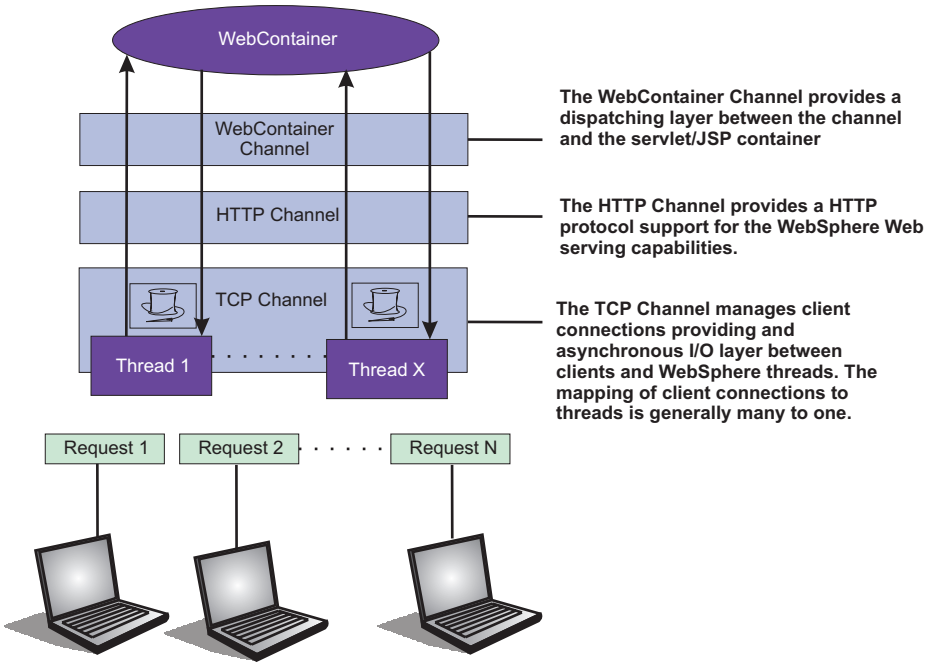


Figure 1. Transport Channel Service

- **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 on 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 allowed, might lead to stalled web sites 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 Web site 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's use. Upon 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, which is adequate for most applications. You should increase the value specified for this parameter if your workload involves a lot of connections and all of these connections can not 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 settings 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 pull-down to assign a particular thread pool to each TCP transport channel. The default settings 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. (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 on **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, CPU usage is usually low and increasing the workload does not increase CPU throughput. Thread dumps show nearly all threads in a call out to the backend resource. If this condition exists, and the backend is tuned correctly, try increasing the minimum number of threads in the pooll until you see improvements in throughput and thread dumps show threads in other areas of the runtime besides the backend call.

   The setting for the Grow as needed parameter should not be 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 on the HTTP transport channel defined for that chain.
  3. Tune HTTP keep-alive.

     The Use persistent (keep-alive) connections setting controls whether or not 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 will never close a socket or where sockets are always proxy or Web servers in front of your application server, a value of -1 will disable the processing which limits the number of requests over a single connection. The persistent timeout will still shutdown some idle sockets and protect 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 should be set to a value that keeps enough connections open so that most clients can obtain a connection available when they need to 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 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 utilize 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 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 need to be changed.
  2. Click on 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 is 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 need to change the value specified for this parameter, make sure the new value enables most requests to be written out in a single write. To determined 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 need 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. Specify one of the following parameters in the Generic JVM arguments field.
4. Click **Apply** or **OK**.
5. 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 is allowed to 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.

Default:	4
Recommended:	Any non-negative integer value is allowed. In practice an integer between 2 and 8 have shown the best performance results.
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 CPU to other threads after a set number of attempts to take a request from the buffer. Typically a lower number of attempts is preferable.

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. This value allows 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.

Default:	4
Recommended:	Any non-negative integer value is allowed. In practice an integer between 2 and 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 CPU to other threads after a set number of attempts to put a request into the buffer. Typically a lower number of attempts is preferable.

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.

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.

Chapter 9. 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

The following parameters include considerations for selecting and configuring the hardware on which the application servers run.

- **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.
- **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.
- **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.
- **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 Web sites might require multiple gigabit links.
  - **Recommendation** Make sure that the highest speed is in use on 10/100/1000 Ethernet networks.

Chapter 10. Tuning operating systems

Use this page to determine your operating system and configure tuning specifications.

About this task

The following 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.

1. Determine your operating system.
2. Select your operating system from the related links section.
3. Configure your settings to optimize performance of Websphere Application Server.

Tuning Windows systems

This topic describes how to tune Windows 2000, Windows XP, and Windows 2003 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.

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**
  - **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 and set (and create if necessary) 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.

    Complete the following steps to immediately acknowledge incoming TCP segments on a server that runs a Microsoft® Windows 2000 operating system:
    1. Start the Registry Editor (regedit.exe).
    2. Locate and click the following registry subkey:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\
```

    3. On the Edit menu, click **Add Value**, and create the following registry value:

```
Value name: TcpDelAckTicks
Data type: REG_DWORD
Value data: 0
Quit Registry Editor.
```

    4. Restart your Windows operating system.

    Similarly, 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 2000, 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.

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

- **timeout_timewait parameter**
  - **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 cost 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, the creation of new connections, and a low throughput due to many connections sitting in the TIME_WAIT state.
  - **How to view or set:**

    Issue the following command to set the timeout_timewait parameter to 30 seconds:

    ```
    echo 30 > /proc/sys/net/ipv4/tcp_fin_timeout
    ```
- **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 later.
  - **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 amount of memory to be shared by issuing the following command:

        ```
        kernel.shmall = 2511724800
        ```

  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.

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 =1
    ```
- **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 tuning parameters topic.
  - **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. The ulimit -a command specifies only 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.
    1. 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
       ```
    2. Save the changes.
    3. Restart your AIX system.

4. To verify the result, type the ulimit -a command on the command line. For example, type # ulimit -a.

– **Default value:** For the AIX operating system, the default setting is `2000`.
– **Recommended value:** The value is application dependent. 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).

- **AIX TCP_KEEPIDLE**
  – **Description:** The keepAlive packet ensures that a connection stays in an active/ESTABLISHED state.
  – **How to view or set:** 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
    ```

  – **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 TypesWebSphere 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 TypesWebSphere 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.

   There are several concerns when enabling large pages, which can cause serious events to occur on the machine when large pages are enabled. For more information on AIX large pages, see the ″Considerations for using large pages″ section at the following address: http://publib.boulder.ibm.com/infocenter/pseries/v5r3/index.jsp?topic=/com.ibm.aix.prftungd/doc/prftungd/large_page_ovw.htm.

   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, see the -Xlp64k option in the Tuning Java virtual machines topic.

- **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 Performance: Resources for learning.

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 Sun Hotspot Java virtual machine (JVM). It is important to use the correct tuning parameters with the Sun JVM to utilize its performance optimizing features. Refer to the Chapter 6, “Tuning the IBM virtual machine for Java,” on page 27 topic for more information about tuning the JVM. Also, consider the following parameters that are specific to the Solaris operating system to ensure that WebSphere Application Server has enough resources.

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 **ulimit** command for the syntax of different shells. For the KornShell (ksh) shell use the **ulimit -n 1024** command. Use the **ulimit -a** command to display the current settings. Use the **ulimit -n 2000** command to set the values.

  - **Default value:** None
  - **Recommended value:** 8000
- **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 Web site 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 Chapter 6, “Tuning the IBM virtual machine for Java,” on page 27 topic for details on setting up the JVM on the HP-UX system. 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.

- 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. Refer to DB2 tuning parameters for information about DB2 tuning parameters.

– **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 Performance: Resources for learning.

– **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 5000
    ```
  - **Default value:** `1`
  - **Recommended value:** `5000` milliseconds

• 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 Web site:

  HP-UX Patch Information

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

  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.

Chapter 11. 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.

- **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://*yourhost*/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.
  - (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 Chapter 10, “Tuning operating systems,” on page 55 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. which specifies the length of time to wait before trying to connect to a server that has been marked temporarily unavailable.

    The plug-in marks a server temporarily unavailable if the connection to the server fails. Although a default value is 60 seconds, you might have to lower this value in order to increase throughput under heavy load conditions. Lowering the RetryInterval might help when the IBM HTTP Server is configured to have fewer than 10 threads per process.

    How can lowering the RetryInterval affect throughput? If the plug-in attempts to connect to a particular application server while the application server threads are busy handling other connections, which happens under heavy load conditions, the connection might time out, causing the plug-in to mark the server temporarily unavailable. If the same plug-in process has other connections open to the same server and a response is received on one of these connections, the server is marked again. If there are only a few threads per IBM HTTP Server process, there might not be an established connection to this application server . When this situation occurs, the plug-in must wait for the entire retry interval.

    Note: Although lowering the RetryInterval can improve performance, if all the application servers are running, a low value can have an adverse affect when one of the application servers is down. In this case, each IBM HTTP Server process attempts to connect and fail more frequently, resulting in increased latency and decreased overall throughput.

  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** .

Chapter 12. Tuning WebSphere applications

This topic provides quick links to information about tuning specific WebSphere application types, and the services and containers that support them.

Note: The WebSphere Application Server documentation contains a finite set of tuning topics to which the following table provides links. Installing the documentation plug-ins for additional components, such as Service integration, might add new entries to the information table of contents. The new entries will not be shown in the table. To see the complete set of application tuning topics available in this information center installation, expand **Tuning performance > Tuning WebSphere applications** in the table of contents.

Product architecture and programming model, at a glance

Application serving environment -- See Tuning the application serving environment	WebSphere applications	WebSphere applications
Servers • Application servers • Java virtual machines • Transport channels • Web servers • More server types • Core groups • Workload balancing **Environment** • Hardware • Operating system • Virtual hosts • Variable settings • Shared libraries **System administration** • Administrative clients • Configuration files • Domains (cells, nodes) **Performance tools** • Monitoring • Tuning performance **Troubleshooting tools** • Diagnostic tools • Support and self-help The product subsystems are discussed in the Product architecture. For the most part, they do not depend on the type of applications being deployed	**Services** • Security • Naming • ORB • Transactions **J2EE applications** • Web applications > Sessions • EJB applications **Clients** • Client applications • Web clients • Web services clients • Administrative clients **Web services** • Web services and Service Oriented Architecture (SOA) • Web services security	**J2EE resources** • Data access resources • Messaging resources • Mail, URLs, and more **WebSphere extensions** • ActivitySessions • Application profiling • Asynchronous beans • Dynamic caching • Dynamic and EJB query • Internationalization • Object pools • Scheduler • Startup beans • Work area

Web services

Monitoring the performance of Web services applications

You can monitor the performance of a Web service that is implemented in the WebSphere Application Server using Performance Monitoring Infrastructure (PMI) tooling.

About this task

You can use the Performance Monitoring Infrastructure (PMI) to measure the time required to process Web services requests. To monitor the performance of a Web services application, follow the steps in this task:

1. Enable PMI services in an application server through the administrative console. Select the Web module named webServicesModule in step 7.
2. Monitor performance with Tivoli Performance Monitor In the left pane of the performance view, expand the host and server. Select **Web Services**. Run the Web services client application.

Results

PMI provides detailed statistics that can help you gain clear insight into the runtime behavior and performance of Web services. Performance counters enable you to see key performance data for each individual Web service including:

- The number of requests dispatched to an implementation bean
- The number of requests dispatched with successful replies
- The average time in milliseconds to process full requests
- The average time in milliseconds between receiving the request and dispatching it to the bean
- The average time in milliseconds between the dispatch and receipt of a reply from the bean. This represents the time spent in business logic.
- The average time in milliseconds between the receipt of a reply from a bean to the return of a result to the client
- The average size of the SOAP request
- The average size of the SOAP reply

To read about other Web services PMI counters, see PMI data organization.

What to do next

If you are having problems with your Web services applications, read about problems and solutions in Troubleshooting Web services.

Web services performance best practices

This topic presents best practices for the performance of Web services applications.

Web services are developed and deployed based on standards provided by the Web Services for Java Platform, Enterprise Edition (Java EE) specification and the Java API for XML-Based Web Services (JAX-WS) and Java Architecture for XML Binding (JAXB) programming models, and is the mechanism used to access a Web service. This article explains performance considerations for Web services supported by this specification.

When you develop or deploy a Web service, several artifacts are required, including a Web Services Description Language (WSDL) file. The WSDL file describes the format and syntax of the Web service input and output SOAP messages. When a Web service is implemented in the WebSphere Application Server runtime, the SOAP message is translated based on the Java EE request. The Java EE-based response is then translated back to a SOAP message.

The most critical performance consideration is the translation between the XML-based SOAP message and the Java object. Performance is high for a Web service implementation in WebSphere Application Server, however, application design, deployment and tuning can be improved. See Monitoring the performance of Web services applications for more information about analyzing and tuning Web services.

If you are using a Web service application that was developed for a WebSphere Application Server version prior to Version 6, you can achieve better performance by running the **wsdeploy** command. The **wsdeploy** command regenerates Web services artifact classes to increase the serialization and deserialization performance.

The wsdeploy command is supported by Java API for XML-based RPC (JAX-RPC) applications. The Java API for XML-Based Web Services (JAX-WS) programming model that is implemented by the application server does not support the wsdeploy command. If your Web services application contains only JAX-WS endpoints, you do not need to run the wsdeploy command, as this command is used to process only JAX-RPC endpoints.

Basic considerations for a high-performance Web services application

The following are basic considerations you should know when designing a Web services application:
- Reduce the Web services requests by using a few highly functional APIs, rather than several simple APIs.
- Design your WSDL file interface to limit the size and complexity of SOAP messages.
- Use the document/literal style argument when you generate the WSDL file.
- Leverage the caching capabilities offered for WebSphere Application Server.
- Test the performance of your Web service.

Additional Web services performance features that you can leverage
- In-process optimizations for Web services to optimize the communication path between a Web services client application and a Web container that are located in the same application server process. For details and enabling this feature, see Web services client to Web container optimized communication.
- Access to Web services over multiple transport protocols extends existing Java API for XML-based remote procedure call (JAX-RPC) capabilities to support non-SOAP bindings such as RMI/IIOP and JMS. These alternative transports can improve performance and quality of service aspects for Web services. For more detailed information see RMI-IIOP using JAX-RPC.
- SOAP with Attachments API for Java (SAAJ) Version 1.2 provides a programming model for Web services relative to JAX-RPC. The SAAJ API provides features to create and process SOAP requests using an XML API. SAAJ supports just-in-time parsing and other internal algorithms. For information about SAAJ or Web services programming, see SOAP with Attachments API for Java.

  SAAJ 1.3 provides support for Web services that are developed and implemented based on the Java API for XML Web Services (JAX-WS) programming model.
- The Web services tooling generates higher performance custom deserializers for all JAX-RPC beans. Redeploying a V5.*x* application into the V6 runtime can decrease the processing time for large messages.
- Serialization and deserialization runtime is enhanced to cache frequently used serializers and deserializers. This can decrease the processing time for large messages.
- The performance of WS-Security encryption and digital signature validation is improved because of the use of the SAAJ implementation.

IBM provides considerable documentation and best practices for Web services application design and development that details these items and more.

Tuning Web services security for Version 7.0 applications

Version 6 and later applications

The Java Cryptography Extension (JCE) is integrated into the software development kit (SDK) Version 1.4.*x* and later. This is no longer an optional package. However, the default JCE jurisdiction policy file shipped with the SDK enables you to use cryptography to enforce this default policy. In addition, you can modify the WS-Security configuration options to achieve the best performance for WS-Security protected applications.

About this task

Using the unrestricted JCE policy files

Due to export and import regulations, the default JCE jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only. To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web services security might improve.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 6, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
2. Click **Java SE 6**
3. Scroll down and click **IBM SDK Policy files**.

   The Unrestricted JCE Policy files for the SDK Web site is displayed.
4. Click **Sign in** and provide your IBM intranet ID and password or register with IBM to download the files.
5. Select the appropriate Unrestricted JCE Policy files and then click **Continue**.
6. View the license agreement and then click **I Agree**.
7. Click **Download Now**.

Results

After following these steps, two Java Archive (JAR) files are placed in the JVM `jre/lib/security/` directory.

Example

Using configuration options to tune WebSphere Application Server

When using WS-Security for message-level protection of SOAP message in WebSphere Application Server, the choice of configuration options can affect the performance of the application. The following guidelines will help you achieve the best performance for your WS-Security protected applications.

1. Use WS-SecureConversation when appropriate for JAX-WS applications. The use of symmetric keys with a Secure Conversation typically performs better than asymmetric keys used with X.509.

   Note: The use of WS-SecureConversation is supported for JAX-WS applications only, not JAX-RPC applications.
2. Use the standard token types provided by WebSphere Application Server. Use of custom tokens is supported, but higher performance is achieved with the use of the provided token types.

3. For signatures, use only the exclusive canonicalization transform algorithm. See the W3 Recommendation Web page (http://www.w3.org/2001/10/xml-exc-c14n#) for more information.
4. Whenever possible, avoid the use of the XPath expression to select which SOAP message parts to protect. The WS-Security policies shipped with WebSphere Application Server for JAX-WS applications use XPath expressions to specify the protection of some elements in the security header, such as Timestamp, SignatureConfirmation, and UsernameToken. The use of these XPath expressions is optimized, but other uses are not.
5. Although there are Websphere Application Server extensions to WS-Security that can be used to insert nonce and timestamp elements into SOAP message parts before signing or encrypting the message parts, you should avoid the use of these extensions for improved performance.
6. There is an option to send the base-64 encoded CipherValue of WS-Security encrypted elements as MTOM attachments. For small encrypted elements, the best performance is achieved by avoiding this option. For larger encrypted elements, the best performance is achieved by using this option.
7. When signing and encrypting elements in the SOAP message, specify the order as sign first, then encrypt.
8. When adding a timestamp element to a message, the timestamp should be added to the security header before the signature element. This is accomplished by using the **Strict** or **LaxTimestampFirst** security header layout option in the WS-Security policy configuration.
9. For JAX-WS applications, use the policy-based configuration rather than WSS API-based configuration.

What to do next

In IBM WebSphere Application Server Version 6.1 and later, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security. See Hardware cryptographic device support for Web Services Security for more information.

Tuning Web services security for Version 5.x applications

Version 5.x application

The Java Cryptography Extension (JCE) policy is integrated into the IBM Software Development Kit (SDK) Version 1.4.*x* and is no longer an optional package. However, due to export and import regulations, the default JCE jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only.

About this task

To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services security might improve.

1. AIX Linux Windows For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:
   a. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html.
   b. Click **Java 1.4.2**.

c. Click **IBM SDK Policy files**. The Unrestricted JCE Policy files for SDK 1.4 Web site is displayed.
d. Enter your user ID and password or register with IBM to download the policy files. The policy files are downloaded onto your machine.

2. Solaris HP-UX For WebSphere Application Server platforms using the Sun-based Java SE Development Kit 6 (JDK 6) Version 1.4.2, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:
   a. Go to the following Web site: http://java.sun.com/j2se/1.4.2/download.html.
   b. Click **Other Downloads**.
   c. Locate the JCE Unlimited Strength Jurisdiction Policy Files 1.4.2 information and click **Download**. The policy files are downloaded onto your machine.

Results

After following either of these sets of steps, two Java Archive (JAR) files are placed in the JVM directory.

```
jre/lib/security/
C:\Program Files\ibm\jre\lib\security
```

Service integration

Tuning messaging engines

Use this task to set tuning properties for the service integration environment.

About this task

The service integration environment includes properties that you can set to improve the performance of a messaging engine or the component of the messaging engine that manages the data store. These properties are known collectively as tuning properties. You can set these properties either with the WebSphere Application Server administrative console or by editing the sib.properties file.

Note: Properties set with the administrative console take precedence over properties set in the sib.properties file.

- Set tuning properties by using the administrative console:
  - Set the tuning properties of a messaging engine.
  - Control the memory buffers used by a messaging engine.
- Use the administrative console to tune the data source.
- Set tuning properties for any of the components mentioned above by editing the sib.properties file.

Setting tuning properties of a messaging engine

You can set the tuning properties for a messaging engine to improve its performance.

About this task

You can set the following tuning property for a messaging engine:

sib.trm.retry
The messaging engine to messaging engine connection retry interval, in seconds. The retry interval is the time delay left between attempts to contact neighboring messaging engines with which communications exist. The default retry interval is 30 seconds.

To set the tuning properties for a messaging engine, use the administrative console to complete the following steps.

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.
6. Restart the messaging engine for the changes to take effect.

Controlling the memory buffers used by a messaging engine

Every messaging engine manages two memory buffers that contain messages and message-related data. You can improve the interaction of a messaging engine with its data store by tuning the properties that set the sizes of the two buffers.

About this task

You can set the following properties to improve the interaction of a messaging engine with its data store:

sib.msgstore.discardableDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is best effort nonpersistent. The default value is 320000, which is approximately 320 kilobytes.

The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises both data that is involved in active transactions, and any other best effort nonpersistent data that the messaging engine has neither discarded nor consumed. The messaging engine holds this data entirely within this memory buffer and never writes the data to the data store. When the messaging engine adds data to the discardable data buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. The messaging engine can discard only data that is not involved in active transactions. This behavior enables the messaging engine to discard best effort nonpersistent messages.

Increasing the size of the discardable data buffer allows more best effort nonpersistent data to be handled before the messaging engine begins to discard messages.

If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transactions, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException.

sib.msgstore.cachedDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The default value is 320000, which is approximately 320 kilobytes.

The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.

sib.msgstore.transactionSendLimit
The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100.

Note: The messaging engine uses approximate calculations to manage the data it holds in the memory buffers. Neither of the **DataBufferSize** properties gives an accurate indication of the amount of

memory that the messaging engine consumes in the JVM heap. The messaging engine can consume considerably more heap storage than the **DataBufferSize** properties indicate.

To set the properties of a messaging engine to improve its interaction with its data store, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.

What to do next

Note: When you change any of these properties, the new values do not take effect until you restart the messaging engine.

Tuning the JDBC data source of a messaging engine

The messaging engine needs to have the correct configuration for JDBC data source to achieve messaging performance on a service integration bus.

Before you begin

Consider whether you need to configure the connection pool for the JDBC data source to achieve your requirements for messaging performance.

About this task

The messaging engine uses the connection pool to obtain its connections to the database. With a heavy workload, a messaging engine might require a large number of concurrent connections to avoid delays waiting for connections to become available in the pool. For example, a very heavily loaded messaging engine might need 50 or more connections. Perform the following steps to configure the connection pool to meet your performance requirements:

1. Ensure that the configuration of your relational database management system (RDBMS) permits the number of connections that you require. Refer to the documentation for your RDBMS for more information.
2. Use the administrative console to set the connection pool parameters for your data source. Navigate to **Resources** → **JDBC** → **Data sources** → ***data_source_name*** → **[Additional Properties] Connection pool properties**.
   a. Set the **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10.

      Note: If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.
   b. Set the **Purge policy** to *EntirePool*. This policy enables the connection pool to release all connections when the messaging engine stops.

Setting tuning properties by editing the sib.properties file

Use this task to set tuning properties for the service integration environment by editing the sib.properties file

About this task

You can set tuning properties to improve the performance of components in the service integration environment. The properties you can set are listed in the tables below:

Properties for a messaging engine

sib.trm.retry
The messaging engine to messaging engine connection retry interval, in seconds. The retry interval is the time delay left between attempts to contact neighboring messaging engines with which communications exist. The default retry interval is 30 seconds.

Properties for the component of a messaging engine that manages the data store

sib.msgstore.discardableDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is best effort nonpersistent. The default value is 320000, which is approximately 320 kilobytes.

The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises both data that is involved in active transactions, and any other best effort nonpersistent data that the messaging engine has neither discarded nor consumed. The messaging engine holds this data entirely within this memory buffer and never writes the data to the data store. When the messaging engine adds data to the discardable data buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. The messaging engine can discard only data that is not involved in active transactions. This behavior enables the messaging engine to discard best effort nonpersistent messages.

Increasing the size of the discardable data buffer allows more best effort nonpersistent data to be handled before the messaging engine begins to discard messages.

If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transactions, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException.

sib.msgstore.cachedDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The default value is 320000, which is approximately 320 kilobytes.

The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.

sib.msgstore.transactionSendLimit
The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100.

To set these properties by editing the sib.properties file, perform the following steps:

1. Navigate to the *profile_root*/properties directory, where *profile_root* is the directory in which profile-specific information is stored.
2. If the directory does not contain a sib.properties file, then copy the template sib.properties files from the *app_server_root*/properties directory, where *app_server_root* is the root directory for the installation of WebSphere Application Server.

3. Using a text editor, open the sib.properties file and add the name and value of the property that you want to set. The format is *name=value*. For example `sib.trm.retry=60`

Tuning messaging performance with service integration technologies

To help optimize performance, you can set tuning properties that control the performance of message-driven beans and other messaging applications deployed to use service integration technologies.

About this task

To optimize the performance of messaging with service integration technologies, you can use the administrative console to set various parameters as described in the steps below. You can also set these parameters using the wsadmin tool.

- View the Available Message Count on a destination.

  Viewing the Available Message Count on a destination enables you to determine whether your message consumers are able to cope with your current workload. If the available message count on a given destination is too high, or is increasing over time, consider some of the tuning recommendations in this topic.

  1. Enable AvailableMessageCount statistics for a queue. If you restart the administrative server, you need to enable AvailableMessageCount statistics again because such runtime settings are not preserved when the server is restarted.
     a. In the navigation pane, click **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)**.
     b. In the content pane, click ***server_name***.
     c. Click the Runtime tab.
     d. In the Currently monitored statistic set, click **Custom**.
     e. On the Custom monitoring level panel, click **SIB Service** → **SIB Messaging Engines** → ***engine_name*** → **Destinations** → **Queues** → ***queue_name***.
     f. Select the AvailableMessageCount option.
     g. Click the **Enable** button at the top of the panel.
  2. View the available message count for a queue.
     a. In the navigation pane, click **Monitoring and Tuning** → **Performance Viewer** → **Current activity**.
     b. In the content pane, click ***server_name***.
     c. Click **Performance Modules** → **SIB Service** → **SIB Messaging Engines** → ***engine_name*** → **Destinations** → **Queues** → ***queue_name***.
     d. Click the **View Module(s)** button at the top of the Resource Selection panel, located on the left side. This displays the AvailableMessageCount data in the Data Monitoring panel, located on the right side.

        You can use the Data Monitoring panel to manage the collection of monitoring data; for example, you can use the buttons to start or stop logging, or to change the data displayed as either a table or graph.

- Monitor MDB Thread Pool Size for the Default Message Provider.

  You might experience a performance bottleneck if there are insufficient threads available for the message-driven beans. There is a trade-off between providing sufficient threads to maximize the throughput of messages and configuring excessive threads, which can lead to CPU starvation of the threads in the application server. If you notice that the throughput for express nonpersistent, reliable nonpersistent, or reliable persistent messaging has fallen as a result of increasing the size of the default thread pool, then decrease the size of the thread pool and reassess the message throughput.

  1. View or change the number of threads in the default thread pool for an application server. By default, message-driven beans use the default thread pool.

a. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Thread Pools** → **Default**. By default the Minimum size value is set to 5 and the Maximum size value is set to 20. The best performance is obtained by setting the Maximum size value to the expected maximum concurrency for all message-driven beans. For high throughput using a single message bean, 41 was found to be the optimal Maximum size value.

b. Change the Maximum size value, then click **OK**.

2. Optional: Create your own thread pool. The default thread pool is also used by other WebSphere Application Server components, so you might want to define a separate thread pool for the message-driven beans. This reduces thread contention for the default thread pool.

   a. Click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Additional Properties] Thread Pools**.

   b. Create a new thread pool.

   c. Create sufficient threads to support the maximum amount of concurrent work for the message-driven beans.

   d. Change the SIB JMS Resource Adapter to use the new thread pool:

      1) Click **Resources** → **Resource Adapters** → **Resource adapters**.

      2) If you cannot see any SIB JMS Resource Adapter instances in the list, expand **Preferences** and enable **Show built-in resources**.

      3) Select the **SIB JMS Resource Adapter** with the appropriate scope depending upon the scope of the connection factories.

      4) Add the name of the new thread pool in the **Thread pool alias** box.

      5) Click **Apply** .

3. Save your changes to the master configuration.

- Tune MDB performance with the default messaging provider.

  1. Click **Resources** → **JMS** → **Activation specifications** → ***activation_specification_name***.

  2. Set the maximum batch size for this activation specification.

     Delivering batches of messages to each MDB endpoint can improve performance, particularly when used with Acknowledge mode set to `Duplicates-ok auto-acknowledge`. However, if message ordering must be retained across failed deliveries, set this parameter to 1.

  3. Set the maximum number of concurrent endpoints for this activation specification.

     The maximum concurrent endpoints parameter controls the amount of concurrent work that can be processed by a message bean. The parameter is applicable to message-driven beans using an activation specification. Increasing the number of concurrent endpoints can improve performance but can increase the number of threads in use at one time. To benefit from a change in this parameter, there should be sufficient threads available in the MDB thread pool to support the concurrent work. However, if message ordering must be retained across failed deliveries, set this parameter to 1.

  4. Save your changes to the master configuration.

  For additional information about tuning the throttling of message-driven beans, including controlling the maximum number of instances of each message bean and the message batch size for serial delivery, see Configuring MDB throttling on the default messaging provider.

- Ensure that application servers hosting one or more messaging engines are provided with an appropriate amount of memory for the message throughput you require.

  You can tune the initial and maximum Java Virtual Machine (JVM) heap sizes when adding a server to a messaging bus, that is when you create a messaging engine. You have the option to do this in any of the following cases:

  – When adding a single server to a bus
  – When adding a cluster to a bus
  – When adding a new server to an existing cluster that is itself a bus member

For an application server that is a bus member of at least one bus, or a member of a cluster that is a bus member of at least one bus, the recommended initial and maximum heap sizes are 768MB.

When you are adding a cluster to a bus, you are recommended to increase the initial and maximum JVM heap sizes for every server in the cluster to 768MB.

– Increasing the initial heap size improves the performance for small average message sizes
– Increasing the maximum heap size improves the performance for higher average message sizes

- Reduce the occurrence of OutOfMemoryError exceptions

  If the cumulative size of the set of messages being processed within a transaction by the service integration bus is large enough to exhaust the JVM heap, OutOfMemoryError exceptions occur. Consider one of the following options for reducing the occurrence of OutOfMemoryError exceptions when processing a large set of messages within a transaction.

  – Increase the JVM heap sizes for the application server.
  – Reduce the cumulative size of the set of messages being processed within the transaction.

- Change the maximum connections in a connection factory for the default messaging provider.

  The maximum connections parameter limits the number of local connections. The default is 10. This parameter should be set to a number equal to or greater than the number of threads (enterprise beans) concurrently sending messages.

  1. Click **Resources** → **JMS** → **Topic connection factories** → ***factory_name*** → **[Additional Properties] Connection pool properties**.
  2. Enter the required value in the **Maximum connections** field.
  3. Click **Apply**.
  4. Save your changes to the master configuration.

- Tune reliability levels for messages.

  The reliability level chosen for the messages has a significant impact on performance. In order of decreasing performance (fastest first), the reliability levels are:

  1. Best-Effort Nonpersistent
  2. Express Nonpersistent
  3. Reliable Nonpersistent
  4. Reliable Persistent
  5. Assured Persistent

  For MDB point-to-point messaging, best-effort nonpersistent throughput is more than six times greater than assured persistent. For more information about reliability levels, see Message reliability levels.

Tuning messaging engine data stores

Obtain an overview of improving the performance of messaging engine data stores.

About this task

- “Tuning the JDBC data source of a messaging engine” on page 76
- “Controlling the memory buffers used by a messaging engine” on page 75
- Sharing connections to benefit from one-phase commit optimization

Tuning the JDBC data source of a messaging engine

The messaging engine needs to have the correct configuration for JDBC data source to achieve messaging performance on a service integration bus.

Before you begin

Consider whether you need to configure the connection pool for the JDBC data source to achieve your requirements for messaging performance.

About this task

The messaging engine uses the connection pool to obtain its connections to the database. With a heavy workload, a messaging engine might require a large number of concurrent connections to avoid delays waiting for connections to become available in the pool. For example, a very heavily loaded messaging engine might need 50 or more connections. Perform the following steps to configure the connection pool to meet your performance requirements:

1. Ensure that the configuration of your relational database management system (RDBMS) permits the number of connections that you require. Refer to the documentation for your RDBMS for more information.
2. Use the administrative console to set the connection pool parameters for your data source. Navigate to **Resources** → **JDBC** → **Data sources** → ***data_source_name*** → **[Additional Properties] Connection pool properties**.
   a. Set the **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10.

      Note: If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.
   b. Set the **Purge policy** to *EntirePool*. This policy enables the connection pool to release all connections when the messaging engine stops.

Controlling the memory buffers used by a messaging engine

Every messaging engine manages two memory buffers that contain messages and message-related data. You can improve the interaction of a messaging engine with its data store by tuning the properties that set the sizes of the two buffers.

About this task

You can set the following properties to improve the interaction of a messaging engine with its data store:

sib.msgstore.discardableDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the quality of service attribute is best effort nonpersistent. The default value is 320000, which is approximately 320 kilobytes.

The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises both data that is involved in active transactions, and any other best effort nonpersistent data that the messaging engine has neither discarded nor consumed. The messaging engine holds this data entirely within this memory buffer and never writes the data to the data store. When the messaging engine adds data to the discardable data buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. The messaging engine can discard only data that is not involved in active transactions. This behavior enables the messaging engine to discard best effort nonpersistent messages.

Increasing the size of the discardable data buffer allows more best effort nonpersistent data to be handled before the messaging engine begins to discard messages.

If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transactions, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException.

sib.msgstore.cachedDataBufferSize
The size in bytes of the data buffer that the messaging engine uses to contain data for which the

quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The default value is 320000, which is approximately 320 kilobytes.

The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.

sib.msgstore.transactionSendLimit
The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100.

Note: The messaging engine uses approximate calculations to manage the data it holds in the memory buffers. Neither of the **DataBufferSize** properties gives an accurate indication of the amount of memory that the messaging engine consumes in the JVM heap. The messaging engine can consume considerably more heap storage than the **DataBufferSize** properties indicate.

To set the properties of a messaging engine to improve its interaction with its data store, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → ***bus_name*** → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.

What to do next

Note: When you change any of these properties, the new values do not take effect until you restart the messaging engine.

Increasing the number of data store tables to relieve concurrency bottleneck

Service integration technologies enables users to spread the data store for a messaging engine across several tables. In typical use this is unlikely to have a significant influence. However, if statistics suggest a concurrency bottleneck on the *SIBnnn* tables for a data store, you might try to solve the problem by increasing the number of tables.

About this task

For more information on the set of tables in a data store see Data store tables

SIB000	contains information about the structure of the data in the other two tables – the “stream table”
SIB001	contains persistent objects – the “permanent item table”
SIB002	contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement – the “temporary item table”

Having multiple tables means you can relieve any performance bottleneck you might have in your system. You can modify *SIBnnn* tables of the data store of a messaging engine. You can increase the number of permanent and temporary tables (*SIB001* and *SIB002*), although there is no way to increase the number of stream tables (*SIB000*).

Example

This example illustrates what the *SIBnnn* tables for a data store might look like after modification:

SIB000	contains information about the structure of the data in the other two tables – the “stream table”
SIB001	contains persistent objects – the “permanent item table”
SIB002	contains persistent objects – the “permanent item table”
SIB003	contains persistent objects – the “permanent item table”
SIB004	contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement – the “temporary item table”
SIB005	contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement – the “temporary item table”

For instructions on how to configure the data store to use multiple item table, see the following topics:

One-phase commit optimization tuning

If you have configured your messaging engine to use a data store, you can achieve better performance by configuring both the messaging engine and container-managed persistent (CMP) beans.

About this task

You need to configure both the CMP bean and the messaging engine resource authorization so that they share the same data source.

1. Open the administrative console.
2. Click **Applications** → **Application Types** → **WebSphere enterprise applications** → ***application_name*** → **[Enterprise Java Bean Properties] Map data sources for all 2.x CMP beans**.
3. On the content pane, select the check boxes next to all the CMP beans.
4. Select *Per application* in the **Resource authorization** selection list.
5. Modify the messaging engine’s resource authorization to *Per application* by modifying the property file `sib.properties` and adding the custom property `sib.msgstore.jdbcResAuthForConnections=Application`.

Setting tuning properties for a mediation

Use this task to tune a mediation for performance by using the administrative console.

Before you begin

Review the guidance on when it is appropriate to tune a mediation for performance in the topic Guidance for tuning mediations for performance.

About this task

You can set the following tuning property in the administrative console to improve the performance of a mediation:

sib:SkipWellFormedCheck
Whether you want to omit the well formed check that is performed on messages after they have been processed by the mediation. Either true or false.

Note: This property is overridden for messages that have the delivery option assured persistent, and a well formed check is always performed.

To set, or unset, one or more tuning properties for a mediation, use the administrative console to complete the following steps:

1. Display the mediation context information:
   a. Click **Service integration** → **Buses** → ***bus_name*** → **[Destination resources] Mediations**.
   b. In the content pane, select the name of the mediation for which you want to configure tuning information.
   c. Click **[Additional Properties] Context information**.
2. In the content pane, click **New**.
3. Type the name of the property in the **Name** field.
4. Select the type `Boolean` in the list box.
5. Type **true** in the **Context Value** field to set the property, or type **false** to unset the property.
6. Click **OK**.
7. Save your changes to the master configuration.

Enabling CMP entity beans and messaging engine data stores to share database connections

Use this task to enable container-managed persistence (CMP) entity beans to share the database connections used by the data store of a messaging engine. This has been estimated as a potential performance improvement of 15% for overall message throughput, but can only be used for entity beans connected to the application server that contains the messaging engine.

About this task

To enable CMP entity beans to share the database connections used by the data store of a messaging engine, complete the following steps.

1. Configure the data store to use a data source that is not XA-capable. For more information about configuring a data store, see Configuring a JDBC data source for a messaging engine.
2. Select the Share data source with CMP option.

   This option is provided on the JMS connection factory or JMS activation specification used to connect to the service integration bus that hosts the bus destination that is used to store and process messages for the CMP bean.

   For example, to select the option on a unified JMS connection factory, complete the following steps:
   a. Display the default messaging provider. In the navigation pane, click **Resources** → **JMS** → **JMS providers**.
   b. Select the default provider for which you want to configure a unified connection factory.
   c. Optional: Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
   d. In the content pane, under Additional Properties, click **Connection factories**
   e. Optional: To create a new unified JMS connection factory, click **New**.

      Specify the following properties for the connection factory:

      Name Type the name by which the connection factory is known for administrative purposes.

      JNDI name
      Type the JNDI name that is used to bind the connection factory into the name space.

Bus name

Type the name of the service integration bus that the connection factory is to create connections to. This service integration bus hosts the destinations that the JMS queues and topics are assigned to.

f. Optional: To change the properties of an existing connection factory, click one of the connection factories displayed. This displays the properties for the connection factory in the content pane.

g. Select the check box for the Share data source with CMP field

h. Click **OK**.

i. Save your changes to the master configuration.

The JMS connection factory can only be used to connect to a "local" messaging engine that is in the application server on which the CMP beans are deployed.

3. Deploy the CMP beans onto the application server that contains the messaging engine, and specify the same data source as used by the messaging engine. You can use the administrative consoles to complete the following steps:

   a. Optional: To determine the data source used by the messaging engine, click **Servers** → **Server Types** → **WebSphere application servers** → ***server_name*** → **[Server messaging] Messaging engines** → ***engine_name*** → **[Additional Properties] Message store** The **Data source name** field displays the name of the data source; by default:

   `jdbc/com.ibm.ws.sib/`*`engine_name`*

   b. Click **Applications** → **New Application** → **New Enterprise Application**.

   c. On the first Preparing for application install page, specify the full path name of the source application file (.ear file otherwise known as an EAR file), then click **Next**

   d. On the second Preparing for application install page, complete the following steps:

      1) Select the check box for the Generate Default Bindings property. Data source bindings (for EJB 1.1 JAR files) are generated based on the JNDI name, data source user name password options. This results in default data source settings for each EJB JAR file. No bean-level data source bindings are generated.

      2) Under Connection Factory Bindings, click the check box for the **Default connection factory bindings:** property, then type the JNDI name for the data source and optionally select a **Resource authorization** value.

      3) Click **Next**

4. If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for Step: Provide default data source mapping for modules containing 1.x entity beans, specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.

5. If your application has CMP beans that are based on the EJB 1.x specification, for Step: Map data sources for all 1.x CMP, specify a JNDI name for data sources to be used for each of the 1.x CMP beans. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error displays after you click **Finish** (step 13) and the installation is cancelled.

6. Complete other panels as needed.

7. On the Summary panel, verify the cell, node, and server onto which the application modules will install:

   a. Beside Cell/Node/Server, click the **Click here** link.

   b. Verify the settings on the Map modules to servers page displayed. Ensure that the application server that is specified contains the messaging engine and its data store.

   c. Specify the Web servers as targets that will serve as routers for requests to this application. This information is used to generate the plug-in configuration file (plugin-cfg.xml) for each Web server.

   d. Click **Finish**.

Results

For more information about installing applications, see Installing application files with the console.

Tuning bus-enabled Web services

You can use the administrative console or a Jacl script to tune performance settings for service integration bus-enabled Web services.

About this task

Bus-enabled Web services dynamically use a fast-path route through the bus where possible. This fast-path route is used if the following criteria are met:

- The inbound port and outbound port for the service are on the same server.
- There are no mediations on the path from the inbound port to the outbound port.

Further optimizations can be made, if your configuration also meets the following criteria:

- The inbound template WSDL URI is the same location as the Outbound Target Service WSDL location URI.
- The inbound service template WSDL service name matches the outbound WSDL service name.
- The inbound service template port name matches the outbound WSDL port name.
- The mapping of the namespaces is disabled (that is, you have set the inbound service property **com.ibm.websphere.wsgw.mapSoapBodyNamespace** to `false`).
- Operation-level security is not enabled on the outbound service.

If your Web services use the fast-path route, you need not tune mediations or the service integration bus. However it is good practise to do so, because a typical environment will have at least one non-fast-path (for example, mediated) service.

To improve the performance of bus-enabled Web services you can tune the following parameters:

- The Java virtual machine heap size. This helps ensure there is enough memory available to process large messages, or messages with large attachments.
- The maximum number of instances of a message-driven bean that are permitted by the activation specification for the service integration technologies resource adapter. This throttles the number of concurrent clients serviced.
- The maximum batch size for batches of messages to be delivered to a client. By default, only a single message is delivered to a message-driven bean instance at one time; you can improve performance by allowing messages to be sent in batches to a message-driven bean.
- The number of threads available to service requests for each client. That is, the number of threads available in the default thread pool, the Web container thread pool and the mediation thread pool for a given application server.
- The number of threads available in the mediation thread pool. This assumes that your mediations use concurrent support where appropriate, as explained in Concurrent mediations.

If you have mediations that act on SOAP headers, you can improve performance by inserting the associated header schemas (`.xsd` files) into the SDO repository.

To tune bus-enabled Web services, complete one of the following two steps:

- Use the administrative console to tune bus-enabled Web services, or
- Use a Jacl script to tune bus-enabled Web services.

If you have mediations that act on SOAP headers, also complete the following step:

- Insert the header schemas into the SDO repository.

- Optional: To use the administrative console to tune bus-enabled Web services, complete the following steps:
  1. Use the topic Tuning Java virtual machines to set the JVM heap size to a larger value than the default value (256 megabytes). The value should generally be as large as possible without incurring paging.
  2. Use the topic Tuning service integration messaging to tune the maximum number of instances of a message-driven bean, the maximum batch size for batches of messages for a bean, and the number of threads available to service requests for a bean.
  3. Use the topic Tuning the application serving environment to tune the general application serving environment, in particular the size of the Web Container Thread Pool. In a server which is exclusively serving requests to bus-enabled Web services, the default thread pool and the Web Container thread pool should be the same size.
  4. Use the topic Configuring the mediation thread pool to configure the number of threads available to concurrent mediations.
- To use a script to tune bus-enabled Web services, use the wsadmin scripting client to run a script similar to the following example. Whilst the values in this script indicate parameters that you should investigate in terms of performance, you must ensure that you understand the impact that changing these parameters will have on the system, especially in cases where bus-enabled Web services may not be the only work that your application server handles.

```
#-------------------------------------------------------------------
# Bus-enabled Web services WebSphere Tuning Script
#-------------------------------------------------------------------
##
# This script is designed to modify some of the tuning pertinent to a
# bus-enabled Web services deployment.
# In order to tune the config parameters, simply change the values
# provided below. This script assumes that all server names in a
# cluster configuration are unique.
#
# To invoke the script, type:
#   wsadmin -f tuneWAS.py <scope> <id>
#      scope      - 'cluster' or 'server'
#      id         - name of target object within scope (i.e. servername)
#
# Example:
#   wsadmin -f tuneWAS.py server server1
#   wsadmin -f tuneWAS.py cluster WSGWCluster
#
#
#-------------------------------------------------------------------

import sys

AdminConfig.setValidationLevel("NONE")

print "Starting script..."
print "Reaqding config parameters..."

#-------------------------------------------------------------------
# COMMON CONFIG PARAMETERS
# - Adjust these parameters based on the intended target system (Defaults in parentheses)
#-------------------------------------------------------------------
# WebContainer Thread Pool (10,50)

minWebPool=10
maxWebPool=15

# Default Thread Pool - (Multiprotocol MDB) (10,50)
minDefaultPool=10
maxDefaultPool=15

# Mediations Thread Pool (1,5)
```

```
minMediationPool=10
maxMediationPool=15

# HTTP KeepAlive settings (true, 100)
keepAliveEnabled="true"
maxPersistentRequests=-1

# Inactivity Timeouts for thread pools (3500)
inactivity=3500

# JVM properties
minHeap=1280
maxHeap=1280
verboseGC="false"
genericArgs=""

# J2CActivationSpec for the SIB_RA Resource adapter
SIB_RA_maxConcurrency=40
SIB_RA_maxBatchSize=5

# Jav2 Security (false for 5.1 and true for 6.0)
j2Security="false"

# Parallel server startup
parallelStart="false"

#--------------------------------------------
# Check/Print Usage
#--------------------------------------------

def printUsageAndExit():
  print
  print "Usage: wsadmin -f tuneWAS.py <cluster | server> <name>"
  sys.exit(0)

#--------------------------------------------
# Misc Procedures
#--------------------------------------------

def getName(objectid):
  endIndex=objectid.index("(")
  return objectId[0:endIndex]

#--------------------------------------------
# Parse command line arguments
#--------------------------------------------

print "Parsing command line arguments..."

if len(sys.argv)<2:
  printUsageAndExit()
else:
  scope=sys.argv[0]
  print "Scope:    %s" % scope

  if scope=="cluster":
    clustername=sys.argv[1]
    print "Cluster:  %s" % clustername
  elif scope=="server":
    servername=sys.argv[1]
    print "Server:   %s" % servername
  else:
    print "Error: Invalid Argument (%s)" % scope
    printUsageAndExit()

#--------------------------------------------
# Obtain server list
```

```
#--------------------------------------------

print
print "Obtaining server list..."

serverList=[]

if scope=="cluster":
  cluster=AdminConfig.getid("/ServerCluster:%s/" % clustername)
  temp=AdminConfig.showAttribute(cluster , "members")
  memberList=" ".split(temp[1:-1])
  for member in memberList:
    memberName=getName(member)
    serverList.insert(0,AdminConfig.getid("/Server:%s/" % memberName))
else:
  server=AdminConfig.getid("/Server:%s/" % servername)
  serverList.insert(0,server)

#--------------------------------------------
# Print config properties
#--------------------------------------------

print
print "WebSphere configuration"
print "-----------------------"
print ""
print "   Enforce Java2 Security:      %s" % j2Security
print

print "Servers:"
for server in serverList:
  print "   %s" % getName(server)

print
print " Web -------------------------------------------"
print "   Min WebContainer Pool Size:  %s" % minWebPool
print "   Max WebContainer Pool Size:  %s" % maxWebPool
print " JVM -------------------------------------------"
print "   Min JVM Heap Size:           %s" % minHeap
print "   Max JVM Heap Size:           %s" % maxHeap
print "   Verbose GC:                  %s" % verboseGC
print


#--------------------------------------------
# Modify cell parameters
#--------------------------------------------

# Accessing cell based security config
print "Accessing security configuration..."
sec=AdminConfig.list("Security")
attrs=[["enforceJava2Security",j2Security]]
print "Updating secuirty..."
AdminConfig.modify(sec,attrs)

#--------------------------------------------
# Modify server parameters
#--------------------------------------------

for server in serverList:
  servername=getName(server)
  print
  print "Serevr: %s" % servername
  print

  # Accessing server startup config
  print "Accessing server startup configuration..."
```

```
print "Parallel Startup (old/new):  %s/%s"
print % (AdminConfig.showAttribute(server,"parallelStartEnabled") , parallelStart)
attrs=[['parallelStartEnabled' , parallelStart]]
print "Updating server startup..."
print
AdminConfig.modify(server , attrs)

# Accessing web container thread pool config
print "Accessing web container thread pool configuration..."
tpList=AdminConfig.list('ThreadPool',server).splitlines()

webPool=filter(lambda x: re.search("WebContainer" , x) , tpList)[0]
print "ThreadPool MaxSize (old/new):             %s/%s"
print % (AdminConfig.showAttribute(webPool , "maximumSize") , maxWebPool)
print "ThreadPool MinSize (old/new):             %s/%s"
print % (AdminConfig.showAttribute(webPool , "minimumSize") , minWebPool)
print "ThreadPool Inactivity Timeout (old/new):  %s/%s"
print % (AdminConfig.showAttribute(webPool , "inactivityTimeout") , inactivity)
attrs=[["maximumSize" , maxWebPool] , ["minimumSize" ,  minWebPool] ,
       ["inactivityTimeout" , inactivity]]
print "Updating web container thread pool..."
print
AdminConfig.modify(webPool , attrs)

# Accessing default thread pool config
print "Accessing default thread pool configuration..."
tpList=AdminConfig.list("ThreadPool" , server)
webPool=filter(lambda x: re.search("Default" , x) , tpList)[0]
print "ThreadPool MaxSize (old/new):             %s/%s"
print % (AdminConfig.showAttribute(webPool , "maximumSize") , maxDefaultPool)
print "ThreadPool MinSize (old/new):             %s/%s"
print % (AdminConfig.showAttribute(webPool , "minimumSize") , minDefaultPool)
print "ThreadPool Inactivity Timeout (old/new):  %s/%s"
print % (AdminConfig.showAttribute(webPool , "inactivityTimeout") , inactivity)
attrs=[["maximumSize" , maxDefaultPool] , ["minimumSize" , minDefaultPool] ,
       ["inactivityTimeout" , inactivity]]
print "Updating default thread pool..."
print
AdminConfig.modify(webPool , attrs)

# Creating Mediations Thread Pool
print "Creating Mediations thread pool"
me=AdminConfig.list(SIBMessagingEngine)
mtpName="%s-mediationThreadPool" % AdminConfig.showAttribute(me , "name")
tpAttrs=[["name" , mtpName] , ["minimumSize" , minMediationPool] ,
         ["maximumSize" , maxMediationPool]]
print "ThreadPool Name    : %s" % mtpName
print "ThreadPool MaxSize : %s" % maxMediationPool
print "ThreadPool MinSize : %s" % minMediationPool
AdminConfig.create("ThreadPool" , me , tpAttrs , "mediationThreadPool")
print "Mediations Thread Pool Created"
print

# Accessing HTTP keepalive config
print "Accessing HTTP KeepAlive configuration..."
HTTPInbound=AdminConfig.list("HTTPInboundChannel" , server)

http2=filter(lambda x: re.search("HTTP_2" , x) , HTTPInbound)[0]
print "KeepAlive Enabled (old/new):        %s/%s"
print % (AdminConfig.showAttribute(http2 , "keepAlive") ,keepAliveEnabled)
print "Max Persistent Requests (old/new):  %s/%s"
print % (AdminConfig.showAttribute(http2 ,
print "maximumPersistentRequests") , maxPersistentRequests)
attrs=[["keepAlive" , keepAliveEnabled] ,
print ["maximumPersistentRequests" , maxPersistentRequests]]
print "Updating HTTP KeepAlives"
print
```

```
  AdminConfig.modify(http2 , attr)

  # Accessing JVM config
  print "Accessing JVM configuration..."
  jvm=AdminConfig.list("JavaVirtualMachine" , server)
  print "Initial Heap Size (old/new):  %s/%s"
  print % (AdminConfig.showAttribute(jvm , "initialHeapSize") , minHeap)
  print "Maximum Heap Size (old/new):  %s/%s"
  print % (AdminConfig.showAttribute(jvm , "maximumHeapSize") , maxHeap)
  print "VerboseGC Enabled (old/new):  %s/%s"
  print % (AdminConfig.showAttribute(jvm , "verboseModeGarbageCollection") , verboseGC)
  attrs=[["initialHeapSize" , minHeap] , ["maximumHeapSize" , maxHeap],
         ["verboseModeGarbageCollection" , verboseGC]]
  print "Updating JVM..."
  print
  AdminConfig.modify(jvm , attrs)

  # Accessing J2CActivationSpec for the SIB Resource Adapter
  print "Modifying the J2CActivationSpec for the SIB Resource Adapter"
  actSpec=AdminConfig.getid("/J2CActivationSpec:SIBWS_OUTBOUND_MDB/")
  propSet=AdminConfig.showAttribute(actSpec , "resourceProperties").splitlines()

  propSet=propSet[0]

  maxConcurrency=["value" , SIB_RA_maxConcurrency]
  maxConcurrency=[maxConcurrency]

  maxBatchSize=["value" , SIB_RA_maxBatchSize]
  maxBatchSize=[maxBatchSize]

  for propId in propSet:
    if AdminConfig.showAttribute(propId , "name")=="maxConcurrency":
      AdminConfig.modify(propId , maxConcurrency)
      print "Custom property changed : %s" % AdminConfig.showall(propId)
    if AdminConfig.showAttribute(propId , "name")=="maxBatchSize":
      AdminConfig.modify(propId , maxBatchSize)
      print "Custom property changed : %s" % AdminConfig.showall(propId)
  print "J2CActivationSpec modifications complete"

print
print "Script completed..."
print "Saving config..."
AdminConfig.save()
```

- Optional: If you have mediations that act on SOAP headers, insert the associated schemas (`.xsd` files) into the SDO repository as described in “Including SOAP header schemas in the SDO repository.”

Including SOAP header schemas in the SDO repository

About this task

Mediations accessing SOAP headers should ensure that the SOAP header schema is made available to the SDO repository. This simplifies access to the header fields (see Web Services code example) and can provide a significant performance benefit. Normally the schema (`.xsd` file) for a SOAP header is already available to the application developer.

Here is an example of a header (used for routing) that is passed in the SOAP message:

```
<soapenv:Header>
<hns0:myClientToken xmlns:hns0="http://www.ibm.com/wbc">
           <UseRoutingId>true</ UseRoutingId >
           <RoutingID>5</ RoutingID >
       </hns0: myClientToken >
</soapenv:Header>
```

Here is an example of an associated header schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.ibm.com/wbc"
           elementFormDefault="unqualified">
<xs:element name=" myClientToken">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="UseRoutingId" type="xs:string"/>
      <xs:element name="RoutingID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

To insert the schema into the SDO repository, complete the following steps:

1. Create a script that contains the following code:
   - For Jython, create a script called `sdoXSDImport.py`:

     ```
     #
     xsdFile=sys.argv[0]
     xsdKey=sys.argv[1]
     sdoRep=AdminControl.queryNames("*,type=SdoRepository,node=%s" % AdminControl.
     getNode)
     print AdminControl.invoke(sdoRep , importResource([xsdKey , xsdFile]))
     ```

   - For Jacl, create a script called `sdoXSDImport.jacl`:

     ```
     #
     set xsdFile [lindex $argv 0]
     set xsdKey  [lindex $argv 1]
     set sdoRep [$AdminControl queryNames *,type=SdoRepository,node=[$AdminControl
      getNode]]
     puts [$AdminControl invoke $sdoRep importResource [list $xsdKey $xsdFile]]
     ```

   Note: To create an equivalent script for removing a resource from the SDO repository, take a copy of this script and modify the final line as follows:
   - Using Jython:

     ```
     AdminControl.invoke(sdoRep , "removeResource" , [[xsdKey , "false"]])
     ```

   - Using Jacl:

     ```
     $AdminControl invoke $sdoRep removeResource [list $xsdKey false]
     ```

2. Use the wsadmin scripting client to insert the schema into the SDO repository by entering the following command.
   - To use the Jython script:

     ```
     wsadmin -lang jython -f sdoXSDImport.py your_header.xsd your_header_namespace
     ```

   - To use the Jacl script:

     ```
     wsadmin -f sdoXSDImport.jacl your_header.xsd your_header_namespace
     ```

   where
   - *your_header*`.xsd` is the name of the file that contains your header schema.
   - *your_header_namespace* is the target namespace for the header. For example `http://yourCompany.com/yourNamespace`.

Security

Tuning, hardening, and maintaining

After installing WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration.

About this task

The following topics are covered in this section:

- Tuning security configurations You can tune your security configuration to balance performance with function. You can achieve this balance following considerations for tuning general security, Common Secure Interoperability version 2 (CSIv2), Lightweight Directory Access Protocol (LDAP) authentication, Web authentication, and authorization. For more information on tuning security, see “Tuning security configurations.”
- Hardening security configurations Several methods exist that you can use to protect your infrastructure and applications from different forms of attack. For more information on hardening your security, see “Hardening security configurations” on page 98.
- Securing passwords in files Password encryption and encoding can add protect to passwords existing in files. For more information on encoding and encrypting passwords, see “Securing passwords in files” on page 99.

Tuning security configurations

You can tune security to balance performance with function. You can achieve this balance following considerations for tuning general security, Common Secure Interoperability version 2 (CSIv2), Lightweight Directory Access Protocol (LDAP) authentication, Web authentication, and authorization.

About this task

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you can disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your Java Platform, Enterprise Edition (Java EE) roles? These questions are things to consider when designing your security infrastructure.

- Consider the following recommendations for tuning general security.
  - Consider disabling Java 2 security manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk.
  - Consider increasing the cache and token timeout if you feel your environment is secure enough. By increasing these values, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token timeout is the exposure of having a token hacked and providing the hacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

    See the article Authentication cache settings for a list of these properties.
  - Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.
  - Use the wsadmin script to complete the access IDs for all the users and groups to speed up the application startup. Complete this action if applications contain many users or groups, or if applications are stopped and started frequently. WebSphere Application Server maps user and group names to unique access IDs in the authorization table. The exact format of the access ID depends on the repository. The access ID can only be determined during and after application deployment. Authorization tables created during assembly time do not have the proper access IDs. See Commands for the AdminApp object for more information about how to update access IDs.
  - Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the ORB tuning guidelines topic.

- If using SSL, enable the SSL session tracking mechanism option as described in the article, Session management settings.
- In some cases, using the unrestricted Java Cryptography Extension (JCE) policy file can improve performance. Refer to the article, Tuning Web services security.
- Distributing the workload to multiple Java virtual machines (JVMs) instead of a single JVM on a single machine can improve the security performance because there is less contention for authorization decisions.

- Consider the following steps to tune Common Secure Interoperability version 2 (CSIv2).
  - Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.
  - If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
  - Consider putting only an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use identity assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk (*) is used, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.
  - Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.
  - If you are communicating only with WebSphere Application Server Version 5 or higher servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

    Note: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.
- Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.
  1. In the administration console, click **Security > Global security**.
  2. Under User account repository, click the **Available realm definitions** drop-down list, select **Standalone LDAP registry** and click **Configure**.
  3. Select the **Ignore case for authorization** option in the standalone LDAP registry configuration, when case-sensitivity is not important.
  4. Select the **Reuse connection** option.
  5. Use the cache features that your LDAP server supports.
  6. Choose either the IBM Tivoli Directory Server or SecureWay® directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.
  7. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory can increase performance in group membership lookup. However, use **Role** only for group mechanisms.
- Consider the following steps to tune Web authentication.
  - Increase the cache and token timeout values if you feel your environment is secure enough. The Web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials that are already created. A disadvantage of increasing the token timeout is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.

- Enable single sign-on (SSO). To configure SSO, click **Security > Global security**. Under Web security, click **Single sign-on (SSO)**.

  SSO is only available when you configure **LTPA** as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 7.0 and does not support SSO. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. Some situations exist where SSO is not a desirable and you do not want to use it in those situations.
- Disable or enabling the **Web Inbound Security Attribute Propagation** option on the Single sign-on (SSO) panel if the function is not required. In some cases, having the function enabled can improve performance. This improvement is most likely for higher volume cases where a considerable number of user registry calls reduces performance. In other cases, having the feature disabled can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.
- The following two custom properties might help to improve performance when security attribute propagation is enabled:
  - **com.ibm.CSI.propagateFirstCallerOnly**

    When this custom property is set to `true` the first caller in the propagation token that stays on the thread is logged when security attribute propagation is enabled. Without setting this property, all of the caller switches are logged, which can affect performance.
  - **com.ibm.CSI.disablePropagationCallerList**

    When this custom property is set to `true` the ability to add a caller or host list in the propagation token is completely disabled. This function is beneficial when the caller or host list in the propagation token is not needed in the environment.

- Consider the following steps to tune authorization.
  - Map your users to groups in the user registry. Associate the groups with your Java Platform, Enterprise Edition (Java EE) roles. This association greatly improves performance when the number of users increases.
  - Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
  - Judiciously assign security-constraints for servlets. For example, you can use the `*.jsp` URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the `*.jsp`, `*.do`, `*.html` extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.
- Use new tuning parameters when using Java 2 security. The new tuning parameters can improve performance significantly, and introduce a new concept called *Read-only Subject*, which enables a new cache for J2C Auth Subjects when using container-managed auth data aliases. If the J2C auth subject does not need to be modified after it is created, the following new tuning parameters can be used to improve Java 2 Security performance:
  - com.ibm.websphere.security.auth.j2c.cacheReadOnlyAuthDataSubjects=true
  - com.ibm.websphere.security.auth.j2c.readOnlyAuthDataSubjectCacheSize=50 (This is the maximum number of subjects in the hashtable of the cache. Once the cache reaches this size, some of the entries are purged. For better performance, this size should be equal to the number of unique subjects (cache based on uniqueness of user principal + auth data alias + managed connection factory instance) when role-based security and Java 2 security are used together).

- Use new tuning parameters to improve the performance of Security Attribute Propagation. The new tuning parameters can be set through custom properties in the administrative console to reduce the extra overhead of Security Attribute Propagation:
  - com.ibm.CSI.disablePropagationCallerList=true
  - com.ibm.CSI.propagateFirstCallerOnly=true (use if you want to track the first caller only).

Results

You always have a trade off between performance, feature, and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide to tune security, create a benchmark before making any change to ensure that the change is improving performance.

What to do next

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus the benefit of configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Secure Sockets Layer performance tips:

Use this page to learn about Secure Sockets Layer (SSL) performance tips. Be sure to consider that performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance.

The following are two types of Secure Sockets Layer (SSL) performance:
- Handshake
- Bulk encryption and decryption

When an SSL connection is established, an SSL handshake occurs. After a connection is made, SSL performs bulk encryption and decryption for each read-write. The performance cost of an SSL handshake is much larger than that of bulk encryption and decryption.

To enhance SSL performance, decrease the number of individual SSL connections and handshakes.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple Transmission Control Protocol/Internet Protocol (TCP/IP) connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections can be impossible if you cannot upgrade to HTTP 1.1.

Another common approach is to decrease the number of connections (both TCP/IP and SSL) between two WebSphere Application Server components. The following guidelines help to verify the HTTP transport of the application server is configured so that the Web server plug-in does not repeatedly reopen new connections to the application server:
- Verify that the maximum number of keep alives are, at minimum, as large as the maximum number of requests per thread of the Web server (or maximum number of processes for IBM HTTP Server on UNIX). Make sure that the Web server plug-in is capable of obtaining a keep alive connection for every possible concurrent connection to the application server. Otherwise, the application server closes the connection after a single request is processed. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of keep alives, to prevent the keep alive connections from consuming the Web container threads.

  Note: HTTP Transports have been deprecated. For instructions on how to set a maximum keep alive value for channel based configurations, see HTTP transport channel settings.

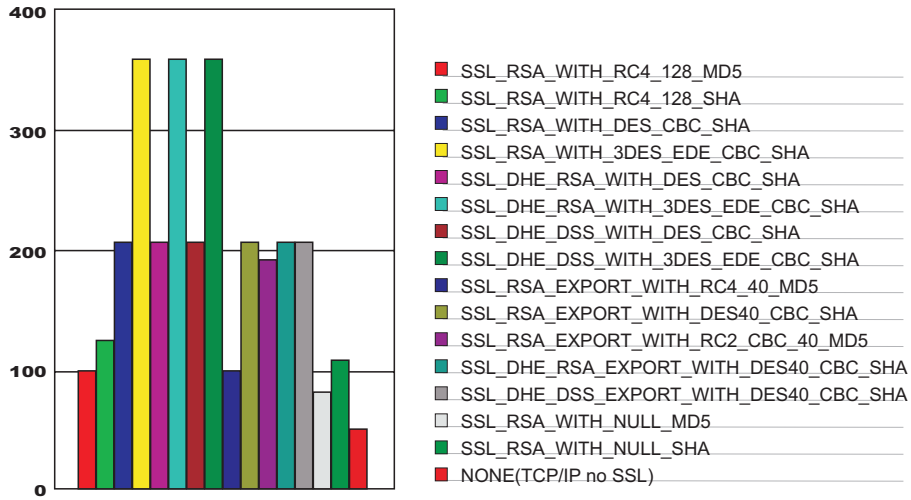
- Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in after 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.
- Use a hardware accelerator if the system performs several SSL handshakes.

  Hardware accelerators currently supported by WebSphere Application Server only increase the SSL handshake performance, not the bulk encryption and decryption. An accelerator typically only benefits the Web server because Web server connections are short-lived. All other SSL connections in WebSphere Application Server are long-lived.
- Use an alternative cipher suite with better performance.

  The performance of a cipher suite is different with software and hardware. Just because a cipher suite performs better in software does not mean a cipher suite will perform better with hardware. Some algorithms are typically inefficient in hardware, for example, Data Encryption Standard (DES) and triple-strength DES (3DES); however, specialized hardware can provide efficient implementations of these same algorithms.

  The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite. The test software calculating the data was Java Secure Socket Extension (JSSE) for both the client and server software, which used no cryptographic hardware support. The test did not include the time to establish a connection, but only the time to transmit data through an established connection. Therefore, the data reveals the relative SSL performance of various cipher suites for long running connections.

  Before establishing a connection, the client enables a single cipher suite for each test case. After the connection is established, the client times how long it takes to write an integer to the server and for the server to write the specified number of bytes back to the client. Varying the amount of data had negligible effects on the relative performance of the cipher suites.



An analysis of the above data reveals the following:
- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion before the WITH identifies the algorithm used only during the SSL handshake.
- MD5 and Secure Hash Algorithm (SHA) are the two hash algorithms used to provide data integrity. MD5 is generally faster than SHA, however, SHA is more secure than MD5.
- DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.

- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Tuning security:

Use the following procedures to tune the performance, without compromising your security settings.

About this task

Enabling security decreases performance. The following tuning parameters provide ways to minimize this performance impact.

- Disable security on any application servers that do not need security. You can disable security in the administrative console by clicking **Security > Global security** and deselecting the **Enable administrative security** option.
- Fine-tune the **Authentication cache timeout** value on the Authentication mechanisms and expiration panel in the administrative console. For more information, see the Global security settings topic.
- Configure the security cache properties. For more information, see the Authentication cache settings topic.
- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console. For more information, see the Session management settings topic.
- Improve the performance of Web services security by downloading a Java Cryptography Extension (JCE) unlimited jurisdiction policy file that does not have restrictions on cryptography strength. For more information, see the “Tuning Web services security for Version 7.0 applications” on page 71 topic.
- Read the Secure Sockets Layer performance tips and “Tuning security configurations” on page 93 topics for more information.

Hardening security configurations

There are several methods that you can use to protect the WebSphere Application Server infrastructure and applications from different forms of attack. Several different techniques can help with multiple forms of attack. Sometimes a single attack can leverage multiple forms of intrusion to achieve the end goal.

About this task

For example, in the simplest case, network sniffing can be used to obtain passwords and those passwords can then be used to mount an application-level attack. The following issues are discussed in IBM WebSphere Developer Technical Journal: WebSphere Application Server V5 advanced security and system hardening:

- Take preventative measures to protect the infrastructure.
- Make applications less vulnerable to attack.
- At a minimum, ensure administrative security is enabled in all WebSphere processes. This protects access to the administrative ConfigService interface and managed beans (MBeans) that enables control over the WebSphere process if it is compromised.
- Ensure Secure Sockets Layer (SSL) is used whenever possible, and mutual SSL whenever possible. However, mutual SSL requires all clients to supply a trusted personal certificate in order to connect.
- Remove any unnecessary certificate authority (CA) signer certificates from your trust stores.
- Change default keystore passwords during or after profile creation using the `AdminTask changeMultipleKeyStorePasswords` command.

- Change your Lightweight Third-Party Authentication (LTPA) keys periodically. By default, this occurs automatically every 12 weeks. If you want to disable this automatic regeneration, remember to manually generate a new set of keys on occasion.
- Common Secure Interoperability version 2 (CSIv2) inbound Basic authentication is supported in this release of WebSphere Application Server. This means that the authentication process is optional. Consider changing the authentication default to 'required'.

Securing passwords in files

Password encoding and encryption deters the casual observation of passwords in server configuration and property files.

About this task

The following topics can be used to add protection for passwords located in files:

- Encoding passwords in files WebSphere Application Server contains some encoded passwords that are not encrypted. The **PropFilePasswordEncoder** utility is included to encode these passwords. For more information on encoding passwords in a file, see "Encoding passwords in files."
- Enabling custom password encryption You need to protect passwords that are contained in your WebSphere Application Server configuration. You can added protection by creating a custom class for encrypting the passwords. For more information on custom password encryption, see "Enabling custom password encryption" on page 102.

Encoding passwords in files:

The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. Use the **PropFilePasswordEncoder** utility to encode passwords stored in properties files. WebSphere Application Server does not provide a utility for decoding the passwords. Encoding is not sufficient to fully protect passwords. Native security is the primary mechanism for protecting passwords used in WebSphere Application Server configuration and property files.

About this task

WebSphere Application Server contains several encoded passwords in files that are not encrypted. WebSphere Application Server provides the **PropFilePasswordEncoder** utility, which you can use to encode passwords. The purpose of password encoding is to deter casual observation of passwords in server configuration and property files. The **PropFilePasswordEncoder** utility does not encode passwords that are contained within XML or XMI files. Instead, WebSphere Application Server automatically encodes the passwords in these files. XML and XMI files that contain encoded passwords include the following:

Table 1. XML and XMI files that contain encoded passwords

File name	Additional information
profile_root/config/cells/*cell_name*/security.xml	The following fields contain encoded passwords: • LTPA password • JAAS authentication data • User registry server password • LDAP user registry bind password • Keystore password • Truststore password • Cryptographic token device password
war/WEB-INF/ibm_web_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture

Table 1. XML and XMI files that contain encoded passwords (continued)

File name	Additional information
ejb jar/META-INF/ibm_ejbjar_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
client jar/META-INF/ibm-appclient_bnd.xml	Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture
ear/META-INF/ibm_application_bnd.xml	Specifies the passwords for the default basic authentication for the run as bindings within all the descriptors
profile_root/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name*/security.xml	The following fields contain encoded passwords: • Keystore password • Truststore password • Cryptographic token device password • Session persistence password
profile_root/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name*/resources.xml	The following fields contain encoded passwords: • WAS40Datasource password • mailTransport password • mailStore password • MQQueue queue mgr password
• *profile_root*/config/cells/*cell_name*/ws-security.xml • *profile_root*/config/cells/*cell_name*/nodes/*node_name*/servers/*server_name*/ws-security	
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

You use the **PropFilePasswordEncoder** utility to encode the passwords in properties files. These files include:

Table 2. The PropFilePasswordEncoder utility - Partial File List

File name	Additional information
profile_root/properties/sas.client.props	Specifies the passwords for the following files: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root/properties/sas.tools.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root/properties/sas.stdclient.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword
profile_root/properties/wsserver.key	

Table 2. The PropFilePasswordEncoder utility - Partial File List (continued)

File name	Additional information
profile_root/profiles/AppSrvXX/properties/sib.client.ssl.properties	Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword
profile_root/UDDIReg/scripts/UDDIUtilityTools.properties	Specifies passwords for: • trustStore.password

To encode a password again in one of the previous files, complete the following steps:

1. Access the file using a text editor and type over the encoded password. The new password is shown is no longer encoded and must be re-encoded.
2. Use the `PropFilePasswordEncoder.bat` or the `PropFilePasswordEncode.sh` file in the *profile_root*`/bin` directory to encode the password again.

   If you are encoding files that are not SAS properties files, type `PropFilePasswordEncoder` *"file_name" password_properties_list*

   Note: When you use the **PropFilePasswordEncoder** utility, a prompt asks whether a backup version of the original file is required. If a backup version is required, a backup file (.bak), is created with the clear text password. Examine the results and then delete this backup file. It contains the unencrypted password. If you do not want to see this prompt, edit the PropFilePasswordEncoder utility and add the following Java system property as a parameter: `-Dcom.ibm.websphere.security.util.createBackup=true` or `-Dcom.ibm.websphere.security.util.createBackup=false`

   A `true` value for the Java system property creates a backup file and a `false` value disables the backup file.

   where:

   "file_name" is the name of the z/SAS properties file, and *password_properties_list* is the name of the properties to encode within the file.

   Note: Only the password should be encoded in this file using the **PropFilePasswordEncoder** tool.

   Use the **PropFilePasswordEncoder** utility to encode WebSphere Application Server password files only. The utility cannot encode passwords that are contained in XML files or other files that contain open and close tags.

Results

If you reopen the affected files, the passwords are encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

PropFilePasswordEncoder command reference:

The **PropFilePasswordEncoder** command encodes passwords that are located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you encode the passwords, a decoding command does not exist.

To encode passwords, you must run this command from the directory:

Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder "file_name" { passwordPropertiesList | -SAS } { -noBackup | -Backup }
    [ -profileName profile ] [ -help | -? ]
```

Parameters

The following option is available for the **PropFilePasswordEncoder** command:

file_name
This required parameter specifies the name of the file in which passwords are encoded.

passwordPropertiesList
This parameter is required if you are encoding passwords in property files other than the sas.client.props file. Specify one or more password properties that you want to encode. The password properties list should be delimited by commas.

-SAS
This parameter is required if you are encoding passwords in the sas.client.props file.

-noBackup
This parameter is optional and the default. The script does not create a backup file. The default value can be altered by adding following Java System Property: `"-Dcom.ibm.websphere.security.util.createBackup=true"`.

-Backup
This parameter is optional. The script creates a backup file, `<file_name>.bak`, which contains passwords in clear text.

-profileName
This parameter is optional. The profile value specifies an application server profile name. The script uses the password encoding algorithm that it retrieves from the specified profile. If you do not specify this parameter, the script uses the default profile.

-help or -?
If you specify this parameter, the script ignores all other parameters and displays usage text.

Enabling custom password encryption:

You need to protect passwords that are contained in your WebSphere Application Server configuration. After creating your server profile, you can added protection by creating a custom class for encrypting the passwords.

Before you begin

Create your custom class for encrypting passwords. For more information, see Plug point for custom password encryption.

About this task

Complete the following steps to enable custom password encryption.

1. Add the following system properties for every server and client process. For server processes, update the `server.xml` file for each process. Add these properties as a genericJvmArgument argument preceded by a **-D** prefix.

   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
   ```

   Note: If the custom encryption class name is com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl, it is automatically enabled when this class is present in the classpath. Do not define the system properties that are listed previously when the custom implementation has this package and class name. To disable

encryption for this class, you must specify `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false` as a system property.

2. Add the Java archive (JAR) file containing the implementation class to the *app_server_root*`/classes` directory so that the WebSphere Application Server runtime can load the file.
3. Restart all server processes.
4. Edit each configuration document that contains a password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point when it is enabled. The `{custom:alias}` tags are displayed in the configuration documents. The passwords, even though they are encrypted, are still Base64-encoded. They seem similar to encoded passwords, except for the tags difference.
5. Encrypt any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. This utility requires that the properties listed previously are defined as system properties in the script to encrypt new passwords instead of encoding them.
6. To decrypt passwords from client Java virtual machines (JVMs), add the properties listed previously as system properties for each client utility.
7. Ensure that all nodes have the custom encryption classes in their class paths prior to enabling this function.

Results

Custom password encryption is enabled.

What to do next

If custom password encryption fails or is no longer required, see “Disabling custom password encryption.”

Disabling custom password encryption:

If custom password encryption fails or is no longer required, perform this task to disable custom password encryption.

Before you begin

Enable custom password encryption.

About this task

Complete the following steps to disable custom password encryption.

1. Change the com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled property to be `false` in the `security.xml` file, but leave the com.ibm.wsspi.security.crypto.customPasswordEncryptionClass property configured. Any passwords in the model that still have the `{custom:alias}` tag are decrypted by using the customer password encryption class.
2. If an encryption key is lost, any passwords that are encrypted with that key cannot be retrieved. To recover a password, retype the password in the password field in plaintext and save the document. The new password must be written out using encoding with the {xor} tag with scripting or from the administrative console.

```
com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
       com.acme.myPasswordEncryptionClass
com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false
```

3. Restart all processes to make the changes effective.

4. Edit each configuration document that contains an encrypted password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point in the presence of the {custom:alias} tag. The {xor} tags display in the configuration documents again after the documents are saved.
5. Decrypt and encode any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. If the encryption class is specified, but custom encryption is disabled, running this utility converts the encryption to encoding and causes the {xor} tags to display again.
6. Disable custom password encryption from the client Java virtual machines (JVMs) by adding the system properties listed previously to all client scripts. This action enables the code to decrypt passwords, but this action is not used to encrypt them again. The {xor} algorithm becomes the default for encoding. Leave the custom password encryption class defined for a time in case any encrypted passwords still exist in the configuration.

Results

Custom password encryption is disabled.

Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See the *Developing and deploying applications* PDF book for a brief description of each WebSphere extension.

Your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java Platform, Enterprise Edition (Java EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application.

Dynamic cache

Tuning dynamic cache with the cache monitor

Use this task to interpret cache monitor statistics to improve the performance of the dynamic cache service.

Before you begin

Verify that dynamic cache is enabled and that the cache monitor application is installed on your application server.

About this task

See the Displaying cache information topic in the *Administering applications and their environment* PDF to configure the cache monitor application.

Use the cache monitor to watch cache hits versus misses. By comparing these two values, you can determine how much dynamic cache is helping your application, and if you can take any additional steps to further improve performance and decrease the cost of processing for your application server.

1. Start cache monitor and click on **Cache Statistics**. You can view the following cache statistics:

Cache statistic	Description
Cache Size	The maximum number of entries that the cache can hold.
Used Entries	The number of cache entries used.
Cache Hits	The number of request responses that are served from the cache.
Cache Misses	The number of request responses that are cacheable but cannot be served from the cache.
LRU Evictions	The number of cache entries removed to make room for new cache entries.
Explicit Removals	The number of cache entries removed or invalidated from the cache based on cache policies or were deleted from the cache through the cache monitor.

2. You can also view the following cache configuration values:

Cache configuration value	Description
Default priority	Specifies the default priority for all cache entries. Lower priority entries are moved from the cache before higher priority entries when the cache is full. You can specify the priority for individual cache entries in the cache policy.
Servlet Caching Enabled	If servlet caching is enabled, results from servlets and JavaServer Pages (JSP) files are cached. See the *Administering applications and their environment* PDF for more information.
Disk Offload Enabled	Specifies if entries that are being removed from the cache are saved to disk. See the *Administering applications and their environment* PDF for more information.

3. Wait for the application server to add data to the cache. You want the number of used cache entries in the cache monitor to be as high as it can go. When the number of used entries is at its highest, the cache can serve responses to as many requests as possible.
4. When the cache has a high number of used entries, reset the statistics. Watch the number of cache hits versus cache misses. If the number of hits is far greater than the number of misses, your cache configuration is optimal. You do not need to take any further actions. If you find a higher number of misses with a lower number of hits, the application server is working hard to generate responses instead of serving the request using a cached value. The application server might be making database queries, or running logic to respond to the requests.
5. If you have a large number of cache misses, increase the number of cache hits by improving the probability that a request can be served from the cache.

   To improve the number of cache hits, you can increase the cache size or configure additional cache policies. See the *Administering applications and their environment* PDF for more information to increase the cache size and to configure cache policies.

Results

By using the cache monitor application, you optimized the performance of the dynamic cache service.

What to do next

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

Chapter 13. 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 parameter hot list 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. Request metrics is another sources of 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 can provide advice on tuning a production system under a real or simulated load.

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

Some critical sections of the application and server code require synchronization to prevent multiple threads from running this code simultaneously and leading to incorrect results. Synchronization preserves correctness, but it can also reduce throughput when several threads must wait for one thread to exit the

critical section. When several threads are waiting to enter a critical section, a thread dump shows these threads waiting in the same procedure. Synchronization can often be reduced by: changing the code to only use synchronization when necessary; reducing the path length of the synchronized code; or reducing the frequency of invoking the synchronized code.

What to do next

Additional references

WebSphere Application Server V6 Scalability and Performance Handbook

WebSphere Application Server Performance Web site

All SPEC jAppServer2004 Results Published by SPEC.

Appendix. Directory conventions

References in product information to *app_server_root*, *profile_root*, and other directories infer specific default directory locations. This topic 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 products or components require multiple locations. Default values for installation actions by root and non-root users are given. If no non-root values are specified, then the default directory values are applicable to both root and non-root users.

app_client_root

The following list shows default installation root directories for the WebSphere Application Client.

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
Non-root	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/AppClient (Java EE Application client only) Windows C:\IBM\WebSphere\AppClient

app_server_root

The following list shows the default installation directories for WebSphere Application Server.

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

cip_app_server_root

A *customized installation package* (CIP) is an installation package created with IBM WebSphere Installation Factory that contains a WebSphere Application Server product bundled with one or more maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

The following list shows the default installation root directories for a CIP where *cip_uid* is the CIP unique ID generated during creation of the build definition file.

User	Directory
Root	AIX /usr/IBM/WebSphere/AppServer/cip/*cip_uid* HP-UX Linux Solaris /opt/IBM/WebSphere/AppServer/cip/*cip_uid* Windows C:\Program Files\IBM\WebSphere\AppServer\cip*cip_uid*
Non-root	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/cip/*cip_uid* Windows C:\IBM\WebSphere\AppServer\cip*cip_uid*

component_root
The component installation root directory is any installation root directory described in this topic. Some programs are for use across multiple components. In particular, the Update Installer for WebSphere Software is for use with WebSphere Application Server, 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. The following list shows the default installation root directory for Version 7 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.

Directory
AIX HP-UX Linux Solaris *product_root*/gsk7 Windows *product_root*\gsk7

if_root This directory represents the root directory of the IBM WebSphere Installation Factory. Because you can download and unpack the Installation Factory to any directory on the file system to which you have write access, this directory's location varies by user. IBM WebSphere Installation Factory is an Eclipse-based tool which creates installation packages for installing WebSphere Application Server in a reliable and repeatable way, tailored to your specific needs.

iip_root
This directory represents the root directory of an *integrated installation package* (IIP) produced by theIBM WebSphere Installation Factory. Because you can create and save an IIP to any directory on the file system to which you have write access, this directory's location varies by user. An IIP is an aggregated installation package that can include one or more generally available installation packages, one or more customized installation packages (CIPs), and other user-specified files and directories.

profile_root
The following list shows the default directory 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*

User	Directory
Non-root	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/AppServer/profiles/ Windows C:\IBM\WebSphere\AppServer\profiles\

plugins_root

The following default installation root is 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
Non-root	AIX HP-UX Linux Solaris *user_home*/IBM/WebSphere/Plugins Windows C:\IBM\WebSphere\Plugins

updi_root

The following list shows the default installation root directories for the Update Installer for WebSphere Software.

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

web_server_root

The following default installation root directories are 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
Non-root	AIX HP-UX Linux Solaris *user_home*/IBM/HTTPServer Windows C:\IBM\HTTPServer

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.

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 is a trademark of Sun Microsystems, Inc. in the United States, other countries, or both.

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