WebSphere® Application Server for z/OS, Version 7.0

Tuning guide

SA23-2252-00

Note

Before using this information, be sure to read the general information under “Notices” on page 123.

Compilation date: September 10, 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 114

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.

  Balancing workloads with clusters

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

  Using the dynamic cache service to improve performance

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

- Ensure your applications perform well.

  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.

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

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.

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. If running Network Deployment, you must enable PMI on both the server and the administrative agent, and restart the server and the administrative agent.
3. Click **Servers** > **Application servers** in the administrative console navigation tree.
4. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.
5. Under the **Configuration** tab, specify the number of processors on the server. This setting is critical to ensure accurate advice for the specific configuration of the system.
6. Select the **Calculation Interval**. PMI data is taken over time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Therefore, details within the advice messages display as averages over this interval.

7. Select the **Maximum Warning Sequence**. The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is updated. For example, if the maximum warning sequence is set to 3, then the advisor sends only three warnings, to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is issued only if the rate of discards exceeds the new threshold setting.
8. Specify **Minimum CPU for Working System**. The minimum central processing unit (CPU) for a working system refers to the CPU level that indicates a application server is under production load. Or, if you want to tune your application server for peak production loads that range from 50-90% CPU utilization, set this value to 50. If the CPU is below this value, some diagnostic and performance advice are still issued. For example, regardless of the CPU level if you are discarding prepared statements at a high rate, you are notified.
9. Specify **CPU Saturated**. The CPU saturated level indicates at what level the CPU is considered fully utilized. The level determines when concurrency rules no longer increase thread pools or other resources, even if they are fully utilized.
10. Click **Apply**.
11. Click **Save**.
12. Click the **Runtime** tab.
13. 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.
14. 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.
15. 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.
16. Click **OK**.
17. 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.
18. 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

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

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.

   If running in a Network Deployment environment, you must enable PMI on both the server and on the administrative agent before restarting the server and the administrative agent.
2. Enable data collection and set the PMI monitoring level to Extended.

   The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server. These monitoring levels and the data selected determine the type of advice you obtain. The performance advisor in 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.

Chapter 4. Tuning index for WebSphere Application Server for z/OS

One of the goals of the WebSphere Application Server for z/OS® programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that WebSphere Application Server for z/OS relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in WebSphere Application Server for z/OS does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won't see any direct use of socket calls or TCP/IP programming in a WebSphere Application Server for z/OS application.

This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF® calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

There are three layers of tuning that need to be addressed:

- "Tuning the z/OS operating system" on page 57
- Chapter 11, "Tuning the WebSphere Application Server for z/OS runtime," on page 65

We deal with these in separate sections under this article. For more information on tuning applications, refer to Using application clients.

Chapter 5. 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.
- **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.
- **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
- **Tuning index**

One of the goals of the product programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that the product relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in the product does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won't see any direct use of socket calls or TCP/IP programming in this application code.

This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

There are four layers of tuning that need to be addressed:
- "Tuning the z/OS operating system" on page 57
- Tuning the subsystems
- Tuning the product runtime
- Tuning for J2EE applications

We deal with the first three in separate sections under this article and briefly touch on the fourth. For more information on tuning applications, refer to Using application clients.

- **Tuning the subsystems**

  Steps involved in tuning the z/OS subsystems to optimize product performance include:
  - DB2 tuning tips for z/OS
  - "Security tuning tips" on page 103
  - Chapter 6, "Tuning TCP/IP buffer sizes," on page 39
  - GRS tuning tips for z/OS
  - Chapter 7, "Tuning the IBM virtual machine for Java," on page 41
  - CICS® tuning tips for z/OS
- **Tuning the product runtime**

  Steps involved in tuning the product runtime to optimize performance include reviewing the:
  - Review the product configuration
  - Internal tracing tips for the product
  - Location of executable programs tips for z/OS
  - "Security tuning tips" on page 103
- **Review the product configuration**

  The first thing to do is review the product configuration. One simple way to do this is to look in your application control and server regions in SDSF. When each server starts, the runtime prints out the current configuration data in the joblog.
- **Internal tracing tips for the product**

  Product traces can be extremely helpful in detecting and diagnosing problems. By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.
  - Verify that you are not collecting more diagnostic data than you need.

    You should check your tracing options to verify that the ras_trace_defaultTracingLevel property is set to `0` or `1`, and that the ras_trace_basic, and ras_trace_detail properties are not set.

    How to view or set: In the administrative console:
    1. Click **Environment > WebSphere variables**.
    2. On the Configuration Tab check for any of these properties in the name field and observe their settings in the value field.
    3. To change the setting for one of these properties, click the property name the name field, and then specify the new setting in the value field. You can also describe the setting in the description field on this tab.
    4. To add one of these properties, click **New**, and then specify the property name the name field, and the property setting in the value field.

– If you use any level of tracing, including ras_trace_defaultTracingLevel=1, verify that the ras_trace_outputLocation property is set to BUFFER.

  When the ras_trace_defaultTracingLevel property is set to `1` exceptions are written to the trace log as well as to the ERROR log.
  - It is best to trace to CTRACE.

    If you are tracing to SYSPRINT with ras_trace_defaultTracingLeve set to `3`, you might experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you might only experience a 15% degradation in throughput.

– Set the ras_trace_BufferCount property to `4` and the ras_trace_BufferSize property to `128`.

  This setting reserves 512 KB of storage for the trace buffers, which is the minimum amount of storage that is allowed, and reduces memory requirements.

– Disable JRAS tracing.

  To disable JRAS tracing, look for the following lines in the trace.dat file that is pointed to by the JVM properties file:

  ```
  com.ibm.ejs.*=all=disable

  com.ibm.ws390.orb=all=disable
  ```

  Verify that both lines are set to `disable`, or delete the two lines.

  Note: If a value is specified for the ras_trace_outputLocation property, you might be tracing and not know it.

- **Location of executable programs tips**

  The next thing to review in the configuration is where your program code is located. IBM recommends that you install as much of the product code in LPA as is reasonable. This ensures that you have eliminated any unnecessary steplibs which can adversely affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD` in the controller and servant procs do not point to any unnecessary libraries. Refer to “UNIX System Services (USS) tuning tips for z/OS” on page 61 for USS shared file system tuning considerations.

  If you choose to not put most of the runtime in LPA, you might find that your processor storage gets a bigger workout as the load increases. At a minimum, the product starts three address spaces, so that any code that is not shared loads three copies rather than one. As the load increases, many more servants might start and contribute additional load on processor storage.

  Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

- **Tuning for J2EE applications**

  Steps involved in tuning the J2EE applications performance include:

  –
  – Chapter 6, “Tuning TCP/IP buffer sizes,” on page 39
  – “Tuning for SOAP” on page 74
  – “Tuning MDB processing on z/OS”

- **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.

Tuning MDB processing on z/OS

To tune MDB processing, you consider and act on a variety of settings together. There is a wide range of values that you can select, and possibilities to consider, because of the variety of workloads possible to run in any given server.

Before you begin

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is “listening in the controller” for these messages. The “listening in the controller” term is used throughout this description of tuning MDB processing.

About this task

If you want to optimize the processing of messages by message-driven beans, use this task to consider and act on the associated settings.

Tuning MDB processing in the server is a part of the greater task of tuning the server’s entire workload. This needs consideration of a variety of settings and the interactions between those settings.

To tune MDB processing, the following settings must be considered together: WLM service class definitions, WebSphere workload profile selection, Message Listener Service Listener Port settings, JMS Connection Factory pooling settings, and WebSphere MQ Queue Manager settings.

It is difficult to give one recommendation about what values to select for each of these settings, given the variety of workloads possible to run in any given server. There are a wide variety of possibilities to take into consideration, including the following considerations:

- The number of message-driven beans.
- The administrative configuration choices, such as whether to map two message-driven beans to the same or different listener ports.
- Different goals regarding the importance of work for message-driven beans relative to other (HTTP, IIOP) types of work running in the server.

The suggested settings below provide a starting point, under the assumption that the server is configured with only a single application consisting of a single message-driven bean installed and running on this server.

The starting point settings are followed by more detailed discussions that explain the rationale behind the suggestions, and describe the listener port function in more detail in the “listening in the controller” case on z/OS. Together they can help you to understand the function and the settings enough to make your own setting selections for your own systems and servers.

1. Set the Listener Port maximum sessions property to at least twice the maximum number of servant worker threads available to the entire server. The value of this property determines the high threshold value (high threshold = maximum sessions) and is used by the throttle to decide when to block or allow requests.
   a. To view this administrative console page, click **Servers-> Application Servers-> *application_server*-> [Communications]-> Messaging-> Message Listener Service-> Listener Ports-> *listener_port***
   b. Set the maximum sessions property to the value you want the MDB throttle to use as its high threshold value.

      The suggested minimum value is computed by the formula:

      ```
      2 * (maximum number of servants) * (number of worker threads in one servant)
      ```

      Here “servants” means the same as “server instances” as the setting is described on the administrative console. To calculate the number of worker threads in a single servant, see the description of “Workload profile”. See the *Developing and deploying applications* PDF for more information.

   For further considerations setting the Listener Port maximum sessions property, see “Concepts and considerations for MDB settings on z/OS” on page 29.

2. Set the WebSphere MQ Queue Connection Factory properties.
   a. To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> WebSphere MQ Queue Connection Factories (under additional properties)**
   b. Select the Connection Factory specified for the Listener Port.
   c. Under the Additional Properties, activate the Connection Pool panel.
   d. Set the Max Connections property for the Connection Pool. Allow one connection for each message-driven bean. This property value could include message-driven beans mapped onto different listener ports, if those listener ports were each, in turn, mapped onto the same connection factory. For more detailed discussions about considerations that affect this setting, see "Concepts and considerations for MDB settings on z/OS."
   e. Under Additional Properties of the Connection Factory, activate the Session Pool panel.
   f. Set the Max Connections property for the Session Pool. Allow one session for each worker thread in a single servant. This property should be set to at least the number of worker threads available to a single servant. For more detailed discussions about considerations that affect this setting, see "Concepts and considerations for MDB settings on z/OS."
3. Set the WebSphere MQ-related properties. Make sure that the backing WebSphere MQ queue manager has been configured with enough resources to support the intended JMS workload coming from WebSphere Application Server (and other clients). In particular, you may want to consider your queue manager's CTHREAD, IDBACK, and IDFORE parameter settings. For more information on these WebSphere MQ settings, see the WebSphere MQ books:

   WebSphere MQ System Setup Guide (SC34-6052-02).

   WebSphere MQ Script Command (MQSC) Reference Book (SC34-6055-03)

   WebSphere MQ Problem Determination Guide (GC34-6054-01).

Example

1. Suppose your server is configured with the maximum server instances value set to 3, (whatever the minimum number may be). Also suppose that your workload profile is LONGWAIT, which means each servant contains 40 worker threads.

   In this case, set your listener port maximum sessions value to at least 240 = 2 * 3 * 40.
2. Suppose that your application contains two individual message-driven beans, each of which has an onMessage() implementation that forwards the message to another JMS destination. Therefore, each message-driven bean needs its own JMS connection factory to perform this task. Suppose the Administrator has mapped each MDB's JMS connection factory resource reference onto the same administratively-defined connection factory used by the listener port that each of these message-driven beans is mapped onto.

   In this case, you need to set the connection factory's Connection Pool Max Connections value to 42. One connection for each of the two message-driven beans to be used by the listener port, and one connection potentially for each of the 40 onMessage() dispatches that night be running concurrently. (Remember that the connection pool is a per-servant pool).
3. Set the connection factory's Session Pool Max Connections to 40, the number of worker threads in a single servant, regardless of the number of servants.

For debugging tips, refer to "MDB throttle support debugging tips" on page 37.

Concepts and considerations for MDB settings on z/OS

To be able to tune the processing of endpoint message-driven beans (MDBs), you need to understand the concepts and considerations for the MDB settings that you configure.

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is "listening in the controller" for these messages. The "listening in the controller" term is used throughout this description of tuning MDB processing.

The following sub-topics provide a set of information that describe the concepts and considerations that you should be aware of to be able to configure MDB settings on z/OS:

- "Basic "listening in the controller" messaging flow"

  When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is "listening in the controller" for these messages. When a message arrives, it flows through a sequence of events.
- "MDB throttle"

  On z/OS, the "MDB throttle" is used to control the amount of work that the server processes at a given time for a message-driven bean. The MDB throttle limits how far the message listener port will "read ahead" to try to ensure that the work request queue does not have a backlog of messages to be processed.
- "MDB throttle settings for message-driven beans on z/OS" on page 31

  You can tune a variety of settings for the "MDB throttle", to control the amount of MDB work that the server processes at a given time.
- "Connection factory settings for message-driven beans on z/OS" on page 33

  You can tune a variety of connection factory settings to control the creation of connections and sessions for MDB work.
- "Message-driven beans, heterogeneous workloads, and workload management on z/OS" on page 36

  A message-driven bean can run on an application server that hosts a heterogeneous workload including other message-driven beans and non-MDB work items. To manage a heterogeneous workload on z/OS, you should use WLM classification and define unique service classes for different-priority work running in the same server.

Basic "listening in the controller" messaging flow

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is "listening in the controller" for these messages. When a message arrives, it flows through a sequence of events.

When a message has arrived on a JMS destination (queue or topic) on which a message-driven bean running is listening, the following sequence of events takes place:

1. The WebSphere MQ JMS queue agent thread (running in the controller) browses the JMS destination. The queue agent identifies the application server on which a message-driven bean is running for that destination. The queue agent calls a Message Reference Handler (MRH) registered by the application server, to tell the server that a message has arrived on a destination on which one of its message-driven beans is listening. A message reference corresponding to this message is sent along with the call to the MRH.
2. If the throttle high threshold value (see below) has not been exceeded, a work request corresponding to the just-browsed message is queued onto the WLM queue in the controller.
3. The work request is dispatched to an individual servant. A ServerSession is obtained from the appropriate ServerSessionPool in that servant and, using the message reference as index, the message is destructively consumed from the JMS destination.
4. The MDB application onMessage(Message) method is dispatched with the consumed message.
5. When the onMessage(Message) method finishes, the servant notifies the controller that the work record has completed.

MDB throttle

On z/OS, the "MDB throttle" is used to control the amount of work that the server processes at a given time for a message-driven bean. The MDB throttle limits how far the message listener port will "read ahead" to try to ensure that the work request queue does not have a backlog of messages to be processed.

The preprocessing, classification, building of and queuing of a work record to prepare for dispatching a specific message-driven bean is a relatively basic operation, especially when compared to the actual business logic of the message-driven bean and the container infrastructure on the application dispatch path. When the rate of messages arriving reaches occasional high volumes, or peaks, the controller can preprocess many more messages than the back-end servants can process by running the associated MDB application. The result of these peaks (in asynchronous work) is that the WLM work request queue becomes backed up waiting for worker threads (in the servants) that have a backlog of messages to be processed.

A backlog of messages to be processed can also occur as a result of situations where the scalable server is taken out of service for a period of time. This causes messages to build up on the JMS destination waiting for the server to restart. When the server does restart, there is a flood of new work introduced into the server.

The MDB throttle limits how far the message listener port will “read ahead” down the JMS queue (or topic), to try to ensure that the work request queue does not have a backlog of messages to be processed.

MDB throttle settings for message-driven beans on z/OS

You can tune a variety of settings for the “MDB throttle”, to control the amount of MDB work that the server processes at a given time.

This topic describes the following concepts and considerations that affect your choice of MDB throttle settings:

MDB throttle – high and low thresholds

Note: The information in this topic is provided on the assumption that you are mapping a single listener port to any given destination. Although the throttle threshold values are calculated individually for each listener port, only a single queue agent thread exists for each destination regardless of the number of listener ports. For example, when listener ports A and B are mapped to queue Q, if listener port B reaches the low threshold, it releases the throttle on the single queue agent thread for queue Q, even though listener port A might have reached its high threshold (and would therefor be blocking if the listener ports were mapped to separate destinations). More specifically, you cannot map multiple listener ports to a single destination and set a high threshold of 1 on each listener port, if you want to perform serial processing on the message-driven beans on each listener port. Because of this restriction, it is strongly recommended that you map a single listener port only to each destination.

The MDB throttle support maintains a count of the current number of in-flight messages for each listener port.

When a message reference is sent to the MRH, as described in “Basic “listening in the controller” messaging flow” on page 30, the in-flight count is incremented by 1. Next, the in-flight count is compared against the high threshold value for this listener port.

- If the in-flight count is less than or equal to the high threshold value, then a work record is queued up onto the WLM queue.
- If the in-flight count exceeds the high threshold value, the queue agent thread that is running the MRH becomes blocked, entering a wait state, and we say that the throttle is “blocking”.

The in-flight count is decremented by 1 whenever the controller is notified that a work record for this Listener Port has completed (whether or not that the application’s transaction was committed). After being

decremented, the in-flight count is checked against the low threshold value for this Listener Port. If the in-flight count drops down to the low threshold value, the previously-blocked queue agent thread is awoken (notified). At that point new work records can be queued onto the WLM queue and we say the throttle has been “released”.

The low threshold and high threshold values are set externally by one setting, the listener port’s Maximum sessions parameter. The high threshold value is set internally equal to the Maximum sessions value defined externally. The low threshold value is computed and set internally by the formula: `low threshold = (high threshold / 2)`, with the value rounding down to the nearest integer.

However, if the Message Listener service has been configured with the Custom Property of MDB.THROTTLE.THRESHOLD.LOW.EQUALS.HIGH defined and set to a value of “true”, then the low threshold value is set internally to the high threshold value (which is the externally-set Maximum sessions property of the listener port).

Note: Please consider the following information, although most users are not be affected by what is described. One queue agent thread is established for each destination, rather than for each listener port. Therefore, it two listener ports are mapped onto the same queue, a throttle-blocking condition on one listener port also results in a blocking of the second listener port’s queueing of work records. The second listener port is blocked even if it has not reached its high threshold value. For simplicity it is recommended that you do not share a destination across more than one listener port.

MDB throttle – tuning considerations

Consider the reason for recommending that you set the listener port “Maximum sessions” value to twice the number of worker threads available in all servants in the scalable server (2*WT). The reason is based on the goal that if you have an available worker thread in some servant, then you do not want to leave it idle because of a blocked MDB throttle. That is, you do not want to have an empty WLM queue, an available servant worker thread, and a blocked throttle.

With the basic recommendation of using the 2*WT value, then, a blocked throttle is released at the moment when the following conditions are true:

- There is one free servant worker thread
- There is nothing on the WLM queue
- There is one message reference browsed but for which a work request was not added to the WLM queue (the throttle blocked instead)

Furthermore, by setting the high threshold to 2*(WT+N) you can ensure that, at the moment a servant worker thread frees up and releases the throttle, there is a backlog of N messages pre-processed and sitting on the WLM queue ready for dispatch. A very high value would introduce the WLM queue overload problem which the throttle was introduced to avoid, but we do not have a specific upper limit to recommend here. (These tuning considerations assume the queue (or topic) is fully-loaded with messages to be processed.)

So, raising the high threshold value allows the server to create a small backlog of preprocessed messages sitting on the WLM queue if a workload spike occurs. The downside or negative of raising the high threshold value is that it increases the chance that a work record for a given message might time out before the application can be dispatched with the given message. (That is, the server might reach the MDB Timeout limit.) The given message is eventually re-delivered to the server, but only later and the processing done up until that time would be wasted. Also, a very large high threshold value would effectively bypass the MDB throttle function, in which case the WLM queue could be overloaded; this would cause the server to fail.

MDB throttle – alternative tuning considerations

Although the scalable server was designed with the goal of maximizing throughput, it is possible to use the listener port settings to achieve other workflow management goals.

For example, a high threshold setting of '1' guarantees that messages are processed in the order that they are received onto the destination.

There might also be other business reasons, based on capacity or other factors, to restrict a particular listener port to much less concurrency than the server would otherwise support. While this is certainly a supported configuration, it might cause the throttle to block when there are idle worker threads available.

MDB throttle example

Suppose your server is configured with the maximum server instances value set to 3, with workload profile of IOBOUND. You have two CPUs, therefore WebSphere Application server will create six worker threads in each servant. Your application (a single MDB mapped to a queue) handles each message relatively quickly (so there is less risk of timeout) and you want the total time from arrival of a given message on your MDB queue until the end of MDB dispatch for this message to be as small as possible.

To provide a quick response time in the case of a surge in work, you opt for a bigger backlog. You set your Listener Port maximum sessions value to `100 = 2 * (3 * 6 + 32)`.

Note: Any value greater than or equal to `36 = 2 * 3 * 6` would keep all available servant worker threads busy. In practice it would probably not be worth the effort to pick the best possible "backlog factor". Therefore, we make a mental estimate after giving the situation some thought, and pick a value so that we end up with the round number, 100.

Connection factory settings for message-driven beans on z/OS

You can tune a variety of connection factory settings to control the creation of connections and sessions for MDB work.

This topic describes the following concepts and considerations that affect your choice of connection factory settings:

Connection factory settings

To attach an application and a server to a particular queue manager with authentication parameters, applications and message listener ports are both bound to connection factories. The server uses the same administrative model to listen for messages arriving for delivery to message-driven beans as the application uses to exploit JMS, in that message listener ports are bound to a queue connection factory or topic connection factory and a corresponding queue or topic.

In addition to specifying the messaging provider settings (for example, queue manager settings) on connection factories, you also specify connection and session pool properties. In particular, for a connection factory, the values of the connection pool property Maximum connections and session pool property Maximum connections must be chosen with different considerations in mind, depending on whether you are using the connection factory for a listener port, for an application, or for both.

Connection pool maximum connections settings

First, in order to avoid waiting for a JMS connection, because of reaching a WebSphere Application Server-defined limit, the connection pool Maximum connections should be set to at least the sum of the following values:

1. One for each message-driven bean that is mapped to any listener port mapped onto this connection factory.
2. The maximum number of connections for one dispatch multiplied by the number of work threads in a servant.

   To determine this maximum connection count, first find the maximum number of connections obtained in a single application dispatch (typically '1'), checking all applications that use this connection factory. Multiply this maximum number of connections by the number of worker threads in a servant. (See the following note for more information about this value.)

Note that only a single servant's worker threads are counted here, because each servant gets its own connection factory with connection and session pools, although the Administrator defines only a single set of property values.

Note:

A message-driven bean might or might not get a JMS connection in performing its application logic in onMessage(). For example, it might get a connection to forward the message to another destination or send a reply, or it might simply update some log and perform no JMS-related calls of its own.

In either case, we need to count 'one' for this message-driven bean in part 1 of the preceding Maximum connection count, as this is the connection used by the listener port. If the message-driven bean onMessage() logic gets one JMS connection, then we then add the number of servant worker threads to the Maximum connection count. If not, then we need not add to the Maximum connection count on behalf of this (MDB) application component.

Of course, other non-MDB application components which use this connection factory to perform JMS calls might cause the administrator to need to include part 2 of the overall Maximum connections count. But, regardless of how many MDB or non-MDB application components use this connection factory, if they each only use one JMS connection per dispatch, then the count in part 2 is only equal to the number of servant worker threads (not the number of applications multiplied by the number of servant worker threads).

Session pool maximum connections settings

This calculation is an easier one to make than it might first appear. For MDB listener port processing in all typical cases, the session pool Maximum connections property should be set to the number of worker threads in a single servant.

The reasons are the fact that JMS sessions are not shared across application dispatches or by the listener port infrastructure, even for clients of the same connection factory, along with the fact that there is a unique session pool for each JMS connection obtained from the connection factory (although the pool property settings are specified only once, at the connection factory level).

It is possible to imagine a case for which the same connection factory is used both by a listener port and an application, with the application having a higher Maximum connections requirement on the session pool setting, but this does not merit further discussion here.

Note: : It is possible to restrict the concurrent MDB work in a server by setting this session pool Maximum connections to less than the number of servant worker threads. This is not recommended. In such a

case, an MDB work request could be dispatched over to a servant, without a session available to process the message. The worker thread at that point would then wait for a session to become available, tying up the valuable worker thread resource.

Should I use a few or many connection factories?

Some users may prefer to keep these calculations simple; for example, by creating a separate connection factory for each message-driven bean (in which case the connection pool Maximum connections property value could simply be set to 1). Others may prefer to manage fewer connection factory administrative objects.

The fact that the connections and sessions used for MDB processing cannot be shared (that is, cannot be used by more than one flow at a time) means that the desire to take advantage of pooling should not be considered a reason for using fewer connection factories. In other words, adding another connection factory does not prevent connection pooling that could otherwise be exploited by MDB listener port processing.

Connection factories – examples

Note:

The scenario:

- Each servant has 12 worker threads. (The number of servants in the server is not important since each servant gets its own pools).
- Listener port LP1 is mapped to connection factory CF1. Message-driven bean MDB1 is mapped to LP1. The onMessage() application code of MDB1 puts a new message onto a forwarding queue, and so MDB1 has a resource reference which is also resolved to CF1.
- Also, in the same server, listener port LP2 is defined and mapped to connection factory CF2. Message-driven beans MDB2A and MDB2B are defined in the same ejb-jar file and are both mapped to LP2 with complementary JMS selectors. The onMessage() application code of MDB2A and MDB2B each does some logging, but neither message-driven bean makes any JMS API calls of its own.

The solution:

- For connection factory CF1, we count one for MDB1. The MDB1 application (which also uses connection factory CF1 to send its forwarding message) uses one JMS connection, for which we count the number of worker threads (12), multiplied by one. Our total connection pool Maximum connections for connection factory CF1, then, is `13 = 12 + 1`.
- For connection factory CF2, we count one for each of MDB2A and MDB2B. There are no applications using CF2, (only the listener port infrastructure), so we set the connection pool Maximum connections for connection factory CF2 equal to 2.
- For each of the two connection factories, the session pool Maximum connections value is set to 12.

Note:

The scenario:

- Again, each servant has 12 worker threads. In this example we only want to use a single connection factory, CF1.
- Each of two listener ports LP1 and LP2 is mapped to connection factory CF1. The message-driven beans MDB1, MDB2, and MDB3 are part of three unique application EAR files. MDB1 is mapped to LP1, but MDB2 and MDB3 are each mapped to LP2.

The solution:

- Up to this point we count that we need three connections for connection factory CF1. However, there is also a servlet component which puts messages on a queue and it uses the same connection factory CF1. So for connection factory CF1, the connection pool Maximum connections setting is `15 = 3 + 12`.

Message-driven beans, heterogeneous workloads, and workload management on z/OS

A message-driven bean can run on an application server that hosts a heterogeneous workload including other message-driven beans and non-MDB work items. To manage a heterogeneous workload on z/OS, you should use WLM classification and define unique service classes for different-priority work running in the same server.

In the set of topics under “Concepts and considerations for MDB settings on z/OS” on page 29, the explanations assume, for simplicity, that your server hosts a single message-driven bean, and that several MDB instances could be simultaneously running on all servant worker threads. Of course, in reality it is likely that any message-driven bean runs on an application server that hosts a heterogeneous workload including other message-driven beans, enterprise beans accessed through IIOP, work such as servlets and JSPs accessed through HTTP, and more work items.

Although the various MDB-related tuning controls provide a way to exert a fine-grained control over the amount of MDB work performed for a given message-driven bean or set of message-driven beans in a given server, we do not recommend using these settings to prioritize certain MDB work above or below other work in the server. Instead, we recommend using WLM classification and defining unique service classes for different-priority work running in the same server. Be sure to allow for at least one servant per unique service class.

This introduces complications in the case that work moving through a single listener port is configured with different selectors mapping onto different service classes. For more information about workload classification, see Classifying z/OS workload.

There is only one throttle for the listener port. This throttle controls the rate at which differently-prioritized messages get queued up as work records on to the WLM queue. The prioritization only occurs with the help of workload management, when the work records are queued up. It does not apply to the phase in which the messages are browsed by the queue agent thread and work records are queued up.

One strategy you can take in such a case is to raise the high threshold value (the value of the Maximum sessions property of the listener port) above the baseline recommendation of “twice the combined number of worker threads in all the server’s servants”. The rationale would be to make sure the WLM queue is loaded enough to allow workload management to decide which work to process next, rather than cutting off the queuing up of work requests. (Cutting off the queuing up of work requests can leave an overabundance of lower-priority work records on the WLM queue when higher priority messages are waiting to be browsed.)

You can develop other strategies beyond what can be discussed here.

Finally, it is also possible that both the following values can be known to some degree of certainty:

- The average number of servant worker threads processing a given message-driven bean
- The average number of available servants (some number between the minimum and maximum is started at any given time)

This could be computed perhaps using PMI, other monitoring tools, or perhaps even by a high-level understanding of how the message-driven bean fits into a greater application flow within a given server.

Should you adjust the baseline formula setting listener port maximum sessions to become twice the number of worker threads available for the *maximum number* of servants in the scalable server instead of twice the actual number of worker threads available in all servants? Although it is hard to give general

recommendations, it may not be worth changing the formula. Lowering the setting introduces the possibility of idle worker threads, and a setting higher than strictly necessary only results in an extra buildup of messages on the WLM queue. The number of extra messages on the queue should still be a small enough number to prevent the serious problem of an overloaded WLM queue causing the server to fail.

MDB throttle support debugging tips

In order to have minimal impact on performance while still collecting needed debugging information for the MDB throttle support, use the following options:

- Trace option: com.ibm.ejs.jms.listener.MessageReferenceListenerPort=all=enabled

  The above trace option does not need to be specified if the MDB trace option is already set.
- System property: com.ibm.mdb.throttle.trace.enabled

  Disabled, if the property is not defined or set to 0.

  Enabled, if the property is set to 1.
- Dynamic trace support for statistics gathering and presentation.

  To enable, disable, or reset messaging statistics, use the following modify command:

  f *<server>*,mdbstats,[enable | disable | reset]

  The response from the console should be:

  ```
  BBOO0211I MODIFY COMMAND MDBSTATS, [ENABLE | DISABLE | RESET] COMPLETED SUCCESSFULLY
  ```

  The following message appears if statistics gathering is not enabled:

  ```
  BBOO0284I  STATISTICS GATHERING NOT ENABLED FOR <string>
  ```

  To display statistics, use the following modify display command:

  f *<server>*,display,work,mdb,stats

  The displayed information per listener port includes:

  NAME The name of the Listener Port for which the statistics are being displayed.

  TIME The amount of time, in seconds, since the stats were enabled or reset.

  TOTAL
  Total number of message references browsed since the stats were enabled or reset.

  IN-FLIGHT
  The current number of in-flight Work Requests.

  EXCS The number of exceptions while queuing the requests.

  BLOCKS
  The total number of instances the Throttle restricts queuing of work requests.

Chapter 6. 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

TCP/IP can be the source of some significant remote method delays. Follow these tips to tune TCP/IP:

Tune the TCP/IP buffer sizes.

1. First, ensure that you have defined enough sockets to your system and that the default socket time-out of 180 seconds is not too high. To allow enough sockets, update the BPXPRMxx parmlib member:
   a. Set MAXSOCKETS for the AF_INET filesystem high enough.
   b. Set MAXFILEPROC high enough.

   We recommend setting MAXSOCKETS and MAXFILEPROC to at least 5000 for low-throughput, 10000 for medium-throughput, and 35000 for high-throughput WebSphere transaction environments. Setting high values for these parameters should not cause excessive use of resources unless the sockets or files are actually allocated.

   Example:

```
/* Open/MVS Parmlib Member                                        */
/* CHANGE HISTORY:                                               */
/*   01/31/02 AEK Increased MAXSOCKETS on AF_UNIX from 10000 to 50000*/
/*                per request from My Developer                  */
/*   10/02/01 JAB Set up shared HFS                              */

/* KERNEL RESOURCES              DEFAULT             MIN MAX         */
/* ========================      =================== === =========== */
   .
   .
   MAXFILEPROC(65535)            /* 64               3   65535       */

   .
   .
    NETWORK DOMAINNAME(AF_INET) DOMAINNUMBER(2) MAXSOCKETS(30000)
   .
```

2. Next check the specification of the port in TCPIP profile dataset to ensure that NODELAYACKS is specified as follows:

   ```
   PORT 8082 TCP NODELAYACKS
   ```

   In your runs, changing this could improve throughput by as much as 50% (this is particularly useful when dealing with trivial workloads). This setting is important for good performance when running SSL.
3. You should ensure that your DNS configuration is optimized so that lookups for frequently-used servers and clients are being cached.

   Caching is sometimes related to the name server's Time To Live (TTL) value. On the one hand, setting the TTL high will ensure good cache hits. However, setting it high also means that, if the Daemon goes down, it will take a while for everyone in the network to be aware of it.

   A good way to verify that your DNS configuration is optimized is to issue the oping and onslookup USS commands. Make sure they respond in a reasonable amount of time. Often a misconfigured DNS or DNS server name will cause delays of 10 seconds or more.

4. Increase the size of the TCPIP send and receive buffers from the default of 16K to at least 64K. This is the size of the buffers including control information beyond what is present in the data that you are sending in your application. To do this specify the following:

   ```
   TCPCONFIG TCPSENDBFRSIZE 65535
             TCPRCVBUFRSIZE 65535
   ```

   Note: It is unreasonable, in some cases, to specify 256K buffers.

5. Increase the default listen backlog. This is used to buffer spikes in new connections which come with a protocol like HTTP. The default listen backlog is 10 requests. We recommend that you increase this value to something larger. For example:

   ```
   protocol_http_backlog=100
   protocol_https_backlog=100
   protocol_iiop_backlog=100
   protocol_ssl_backlog=100
   ```

6. Reduce the finwait2 time. In the most demanding benchmarks you may find that even defining 65K sockets and file descriptors does not give you enough 'free' sockets to run 100%. When a socket is closed abnormally (for example, no longer needed) it is not made available immediately. Instead it is placed into a state called finwait2 (this is what shows up in the netstat -s command). It waits there for a period of time before it is made available in the free pool. The default for this is 600 seconds.

   Note: Unless you have trouble using up sockets, we recommend that you leave this set to the default value.

   If you are using z/OS V1.2 or above, you can control the amount of time the socket stays in finwait2 state by specifying the following in the configuration file:

   ```
   FINWAIT2TIME 60
   ```

Results

What to do next

Chapter 7. 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.
- 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.

Both the controller and the servant contain a JVM. The information contained in this topic applies only to the JVM in the servant. Typically, the JVM in the controller does not need to be tuned.

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.

   To speed up JVM initialization and improve server startup time, specify the following command-line arguments in the **General JVM arguments** field in the General Properties section of the Configuration Tab.

```
-Xquickstart
-Xverify:none
```

3. Make sure the debug version of the JVM libjava_g file is not included in your libpath.
4. Verify the classpath as part of the Java configuration.

   In the General Properties section of the Configuration Tab, enter the classpath in the text box of the Classpath option. Make sure that the classpath points to only the classes you need. If possible, tthe classes that are referenced most frequently should be located near the front of the path.

   Sometimes poor performance is caused by a missing class. The class loader will look in it's tables of already loaded classes and if the class is not found to be already loaded it will search for it. This search process can cause a high amount of I/O activity to the HFS volumes. To determine if this is the problem you can collect CTRACE records from the file system. Once you determine which class is not being found you can repair the problem by providing the class or by removing the need for it.

5. 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.

Java heap information is contained in SMF records and can be viewed dynamically using the DISPLAY,JVMHEAP console command.

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**.

c. Select either **Control** or **Servant**, and then select **Java virtual machine**.

d. 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.

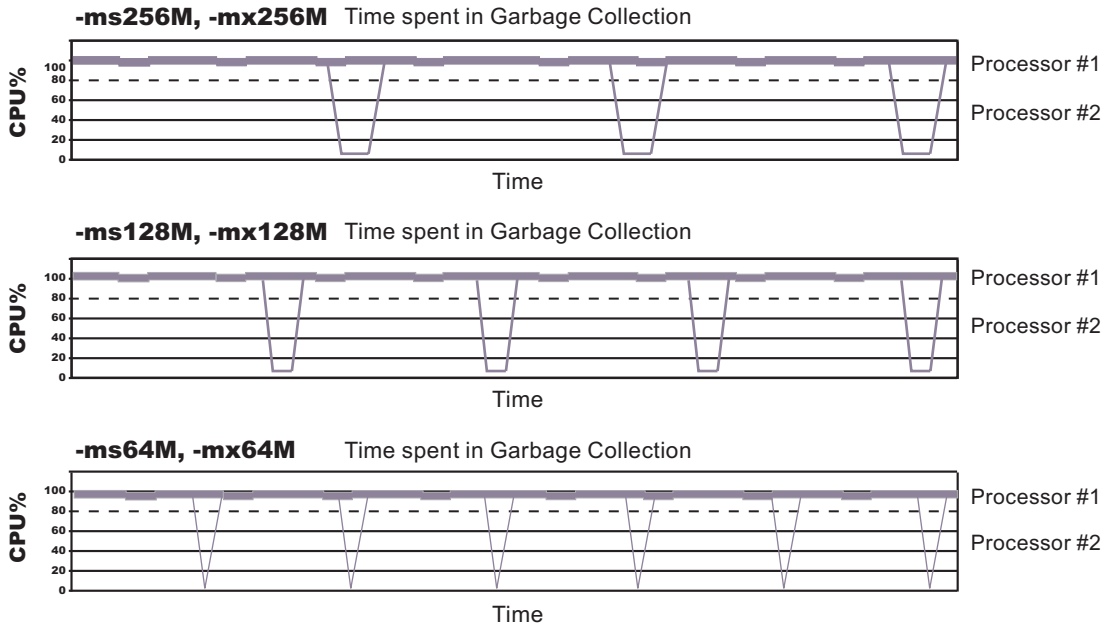
   For performance analysis, the initial and maximum heap sizes should be equal.

   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.

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.

If you have servers configured to run in 64-bit mode, you can specify a JVM maximum heap size for those servers that is significantly larger than the default setting. For example, you can specify an initial maximum heap size of 1844 MB.

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

e. Click **Apply**.

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

g. 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.

6. 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 check if the application is overusing objects, by observing the counters for the JVM runtime. You have to specify the -XrunpmiJvmtiProfiler command-line option, as well as the JVM module maximum level in order 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.

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.

7. Tune garbage collection

   The JVM uses a parallel garbage collector to fully exploit an SMP during most garbage collection cycles.

   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**.
   c. Select either **Control** or **Servant**, and then select **Java virtual machine**.
   d. Enter the –X option you want to change in the **Generic JVM arguments** field.
   e. Click **Apply**.
   f. Click **Save** to save your changes to the master configuration.
   g. 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.

8. **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.

   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**

c. Select either **Control** or **Servant**, and then select **Java virtual machine**.
d. Enter `-Xshareclasses:none` in the **Generic JVM arguments** field.
e. Click **OK**.
f. Click **Save** to save your changes to the master configuration.
g. 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.

9. 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 > Java virtual machine > Custom properties > New**.
   c. Enter `config_consistency_check` in the **Name** field and `false` in the **Value** field.
   d. Click **OK** and then **Save** to apply these changes to the master configuration.
   e. 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 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.

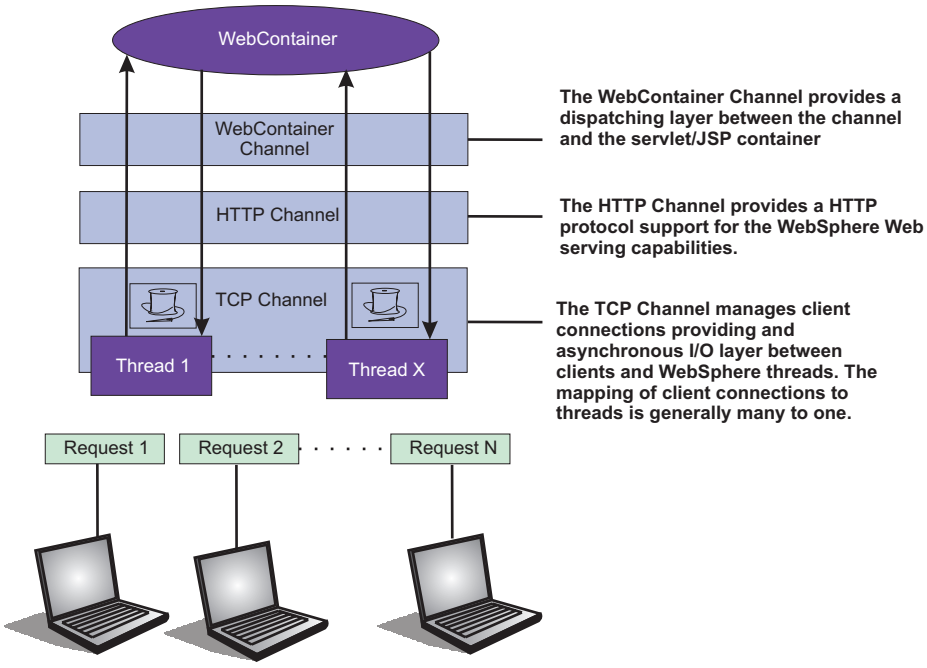


Figure 1. Transport Channel Service

- **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.

- **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.
  - **Description:** The amount of storage required for z/OS is mostly dependent on the number of servers and the size of the Java Virtual Machine (JVM) heap for each server.
  - **Recommendation:** For a single server with 1 GB JVM heap, allocate a minimum of 1GB of 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.
- An **IBM S/390® or zSeries® Model** that supports the software requirement of z/OS V1R2.
- **Storage**
  - Storage requirements are higher than for traditional workloads
  - **Recommendation**
    - Virtual storage default should be about 370 MB per servant, which includes a 256 MB default heap size and a default initial LE heap size of 80 MB.
    - Real storage minimum is 376 MB per LPAR for a light load such as the IVP. For most real-world applications, we recommend 2 GB or higher. We have seen applications that require as much as 8 GB of real to operate at peak load.

- **DASD**
  - **Recommendation**
    - To maximize your performance, we recommend a fast DASD subsystem (for example, IBM Shark), running with a high cache read/write hit rate.
- **Networking**
  - **Recommendation**
    - For high bandwidth applications, we recommend at least a 1 Gb Ethernet connection. If your applications have extremely high bandwidth requirements, you may need additional Ethernet connections.
- **Cryptography**
  - **Recommendation**
    - For applications that make heavy use of cryptography, we recommend the zSeries or S/390 cryptographic hardware and the Integrated Cryptographic Service Facility. For more information, refer to the zSeries and S/390 Cryptography Web site.

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 the z/OS operating system

Use these steps to tune your z/OS operating system to optimize WebSphere Application Server performance.

1. Chapter 12, “Tuning storage,” on page 67
2. “z/OS operating system tuning tips”
3. “UNIX System Services (USS) tuning tips for z/OS” on page 61
4. Chapter 13, “Workload management (WLM) tuning tips for z/OS,” on page 69
5. “Resource Recovery Service (RRS) tuning tips for z/OS” on page 58
6. “Fine tuning the LE heap” on page 62

z/OS operating system tuning tips

There are several configuration changes you can make to z/OS system components that might improve product performance.

You might want to make one or more of the following changes to the indicated z/OS components:

- CTRACE

  The first place to review is your CTRACE configuration. Ensure that all components are either set to MIN or OFF. To display the CTRACE options for all components on your system, issue the following command from the operator console:

  ```
  D TRACE,COMP=ALL
  ```

  To change the setting for an individual component to its minimum tracing value, use the following command, where xxx is the component ID.

  ```
  TRACE CT,OFF,COMP=xxx
  ```

  This configuration change eliminates the unnecessary overhead of collecting trace information that is not needed. Often during debug, CTRACE is turned on for a component and not shut off when the problem is resolved.
- SMF

  Ensure that you are not collecting more SMF data than you need. Review the SMFPRM*xx* settings to ensure that only the minimum number of records are collected.

  Use SMF 92 or 120 only for diagnostics.
  – SMF Type 92

SMF Type 92 records are created each time an HFS file is opened, closed, deleted, and so forth. Almost every Web server request references HFS files, so thousands of SMF Type 92 records are created. Unless you specifically need this information, turn off SMF Type 92 records. In the following example, we have disabled the collection of SMF type 92 records:

Example:

```
ACTIVE,
DSNAME(SYS1.&.SYSNAME..SMF.MAN1;SYS1.&SYSNAME..SMF.MAN2;),
NOPROMPT,
REC(PERM),
MAXDORM(3000),
STATUS(010000),
JWT(0510),
SID(&SYSNAME;(1:4)),
LISTDSN,
SYS(NOTYPE(19,40,92)),
INTVAL(30),
SYNCVAL(00),
SYS(DETAIL,INTERVAL(SMF,SYNC)),
SYS(EXITS(IEFACTRT,IEFUJI,IEFU29,IEFU83,IEFU84,IEFU85,IEFUJV,IEFUSI))
```

– SMF Type 120

You might find that running with SMF 120 records in production is appropriate, because these records give information specific to applications that are running on the product, such as response time for Enterprise Edition (Java EE) artifacts, bytes transferred, and so forth. If you do choose to run with SMF 120 records enabled, we recommend that you use server interval SMF records and container interval SMF records rather than server activity records and container activity records. See the *Troubleshooting and support* PDF for a description of the SMF 120 record.

The *Troubleshooting and support* PDF also describes the steps involved in controlling collection of SMF 120 records. To enable specific record types, specify the following properties:

- server_SMF_server_activity_enabled=0 (or server_SMF_server_activity_enabled = false)
- server_SMF_server_interval_enabled=1 (or server_SMF_server_interval_enabled = true)
- server_SMF_container_activity_enabled=0 (or false)
- server_SMF_container_interval_enabled=1 (or true)
- server_SMF_interval_length=1800
- server_SMF_request_activity_enabled=1 (or true)

- You might also want to review your DB2 records and the standard RMF™ written SMF records, and ensure that the SMF data sets are allocated optimally. DB2 SMF records 100, 101 and 102 affect performance and should be used only for monitoring DB2 performance with the DB2 PM tool. If you are not monitoring DB2 performance, you should consider not collecting those SMF records.

Resource Recovery Service (RRS) tuning tips for z/OS

Use these tips to tune your z/OS operating system to optimize WebSphere Application Server performance.

- For best throughput, use coupling facility (CF) logger for the RRS log.

  DASD logger can limit your throughput because it is I/O-sensitive. The CF logger has much more throughput (in one measurement, the CF logger was six times faster than the DASD logger). Throughput will benefit from moving the RRS logs in logger to a coupling facility (CF) logstream. Doing so will help transactions complete quickly and not require any DASD I/O. If it's not possible to use CF logs, use well performing DASD and make sure the logs are allocated with large CI sizes.

- Ensure that your CF logger configuration is optimal by using SMF 88 records.

  See the tuning section of *z/OS MVS Setting Up a Sysplex* or the chapter on System Logger Accounting in *z/OS MVS System Management Facilities (SMF)* for details. In any case, you should monitor the logger to ensure that there is a sufficient size in the CF and that offloading is not impacting the overall throughput. The transaction logs are shared I/O-intensive resources in the mainline and can affect throughput dramatically if mistuned.

- Set adequate default values for the LOGR policy.

Default values of LOGR policy may have an impact on performance. We recommend the default settings in the table below.

Table 1. Recommended default setting for LOGR

Log Stream	Initial Size	Size
RM.DATA	1 MB	1MB
MAIN.UR	5 MB	50 MB
DELAYED .UR	5 MB	50 MB
RESTART	1 MB	5 MB
ARCHIVE	5 MB	50 MB

- Review XA Resource Managers log sizes.

  If you are using XA Resource Managers and you have chosen to put the logs in the logger, you may have to review the log sizes. As of this writing, we cannot give specific recommendations.

  You can configure the XA logs in the install dialog to live either in the HFS or in logstreams. If you are not using global transactions involving XA resources, there is no point in putting the log in a logstream. If the XA logs are placed in logstreams, we recommend that they be in the Coupling Facility instead of DASD. The default names are 'HLQ.server.M' and 'HLQ.server.D' where HLQ is a user-defined value between 1-8 characters specified in the install dialog, and 'server' is the server short name. It is the installer's responsibility to ensure that the HLQ + server name is unique across the configuration. If it is not, the server will fail to start because the user data in the existing logstream will not match that of the new server. The logs (and structures, if applicable) are created in job 'BBOLOGSA' in the install dialog. If structures need to be allocated, there is also a step indicating what structure names need to be added to the CFRM policy. We recommend 5MB initial and 20MB max sizes for both of these logstreams.
- Eliminate archive log if not needed.

  If you don't need the archive log, we recommend that you eliminate it since it can introduce extra DASD I/Os. The archive log contains the results of completed transactions. Normally, the archive log is not needed. Following is an example of disabling archive logging.

  Example:

```
//STEP1 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)

    DELETE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
    DELETE STRUCTURE NAME(RRSSTRUCT1)
/*
//STEP2 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DEFINE STRUCTURE NAME(RRSSTRUCT1)
           LOGSNUM(9)


    DEFINE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
```

```
            AUTODELETE(YES)
            RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
            STRUCTNAME(RRSSTRUCT1)
            STG_DUPLEX(YES)
            DUPLEXMODE(UNCOND)
            LS_DATACLAS(SYSPLEX)
            LS_STORCLAS(LOGGER)
            HLQ(IXGLOGR)
            AUTODELETE(YES)
            RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
            STRUCTNAME(RRSSTRUCT1)
            STG_DUPLEX(YES)
            DUPLEXMODE(UNCOND)
            LS_DATACLAS(SYSPLEX)
            LS_STORCLAS(LOGGER)
            HLQ(IXGLOGR)
            AUTODELETE(YES)
            RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
            STRUCTNAME(RRSSTRUCT1)
            STG_DUPLEX(YES)
            DUPLEXMODE(UNCOND)
            LS_DATACLAS(SYSPLEX)
            LS_STORCLAS(LOGGER)
            HLQ(IXGLOGR)
            AUTODELETE(YES)
            RETPD(3)
/*

//* DEFINE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)
            //* STRUCTNAME(RRSSTRUCT1)
            //* STG_DUPLEX(YES)
            //* DUPLEXMODE(UNCOND)
            //* LS_DATACLAS(SYSPLEX)
            //* LS_STORCLAS(LOGGER)
            //* HLQ(IXGLOGR)
            //* AUTODELETE(YES)
            //* RETPD(3)
```

Note: RRS manual recommends creating one structure per log stream.

LE tuning tips for z/OS

Enabling xplink in the runtime environment and compiling applications with xplink enabled improves performance in z/OS V1R2.

- For best performance, use the LPALSTxx parmlib member to ensure that LE and C++ runtimes are loaded into LPA, as shown in the following example:

  Example: sys1.parmlib(LPALSTxx):

```
******************************** Top of Data ********************
USER.LPALIB,
ISF.SISFLPA,                            SDSF
CEE.SCEELPA,
                          LANGUAGE ENVIRONMENT
CBC.SCLBDLL,                            C++ RUNTIME
.
.
.

******************************** Bottom of Data *****************
```

- Ensure that the Language Environment® data sets, SCEERUN and SCEERUN2, are authorized to enable xplink.

For processes that run the client ORB, since they start the JVM, must run with xplink (on). For best performance, compile applications that use JNI services with xplink enabled. Compiling applications with xplink enabled improves performance in z/OS V1R2. As you move from z/OS V1R2 to z/OS V1R6 you should experience additional performance improvements when all of the LE services calls are xplink enabled.

- Ensure that you are **NOT** using the following options during production:
  - RPTSTG(ON)
  - RPTOPTS(ON)
  - HEAPCHK(ON)
- Turn LE heappools on.

  If you are running a client on z/OS, setting the following: SET LEPARM='HEAPP(ON)' in a shell script, turns on LE heappools, which should improve the performance of the client.
- Refer to "Fine tuning the LE heap" on page 62

Note: Do not modify LE parameters without consulting IBM support. The LE parameters are set internally to ensure the best possible performance of the WebSphere Application Server, which is the main LE application running in the address space. If you need to add or change LE parameters, make sure that you work with the IBM WebSphere support team to ensure that the internally set parameters are not compromised. The appropriate interface for making these changes is through the `PARM=` parameter of the `EXEC PGM=BPXBATSL` statement in the startup JCL.

UNIX System Services (USS) tuning tips for z/OS

Use these tips to tune your z/OS operating system to optimize WebSphere Application Server performance.

WebSphere Application Server for z/OS no longer requires or recommends the shared file system for the configuration files, since it maintains its own mechanism for managing this data in a cluster. However, WebSphere for z/OS does require the shared files system for XA partner logs. Your application may also use the shared file system. This article provides some basic tuning information for the shared file system.

For basic z/OS UNIX® System Services performance information, refer to the following web site: http://www.ibm.com/servers/eserver/zseries/ebusiness/perform.html

- Mount the shared file system R/O.

  Special consideration needs to be made to file system access when you run in a sysplex. If you mount the file system R/W in a shared file system environment, only one system will have local access to the files. All other systems have remote access to the files which negatively affects performance. You may choose to put all of the files for WebSphere in their own mountable file system and mount it R/O to improve performance. However, to change your current application or install new applications, the file system must be mounted R/W. You will need to put operational procedures in place to ensure that the file system is mounted R/W when updating or installing applications.
- HFS files caching.

  HFS Files Caching Read/Write files are cached in kernel data spaces. In order to determine what files would be good candidates for file caching you can use SMF 92 records.

  Initial cache size is defined in BPXPRMxx.
- Consider using zFS.

  z/OS has introduced a new file system called zFS which should provide improved file system access. You may benefit from using the zFS for your UNIX file system. See *z/OS UNIX System Services Planning* for more information.
- Use the filecache command.

  High activity, read-only files can be cached in the USS kernel using the filecache command. Access to files in the filecache can be much more efficient than access to files in the shared file system, even if

the shared file system files are cached in dataspaces. GRS latch contention, which sometimes is an issue for frequently accessed files shared file system, will not affect files in the filecache.

To filecache important files at startup, you can add filecache command to your /etc/rc file. Unfortunately, files which are modified after being added to the filecache may not be eligible for caching until the file system is unmounted and remounted, or until the system is re-IPLed. Refer to *z/OS UNIX System Services Command Reference* for more information about the filecache command.

Example of using the filecache command:

```
/usr/sbin/filecache -a /usr/lpp/WebSphere/V5R0M0/
MQSeries/java/samples/base/de_DE/mqsample.html
```

Fine tuning the LE heap

Use these steps to tune your z/OS operating system to optimize WebSphere Application Server performance.

About this task

The LE Heap is an area of storage management to be concerned with. For servers, IBM has compiled default values for HEAP and HEAPPOOL into the server main programs. These are good starting points for simple applications. To fine tune the LE Heap settings, use the following procedure:

1. Generate a report on storage utilization for your application servers. Use the LE function RPTSTG(ON) in a SET LEPARM= statement in JCL as shown in the example:.

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON)'
   ```

   Results appear in servant joblog.
2. To bring the server down cleanly, use the following VARY command:

   ```
   VARY WLM,APPLENV=xxxx,QUIESCE
   ```

   The following example shows the servant SYSPRINT DD output from the `RPTSTG(ON)` option.

   Example:

   ```
   .   .   .
   0    HEAP statistics:
          Initial size:                                  83886080

          Increment size:                                 5242880
          Total heap storage used (sugg. initial size):  184809328

          Successful Get Heap requests:                    426551
          Successful Free Heap requests:                   424262
          Number of segments allocated:                         1
          Number of segments freed:                             0
     .   .   .

    Suggested Percentages for current Cell Sizes:
      HEAPP(ON,8,6,16,4,80,42,808,45,960,5,2048,20)
    Suggested Cell Sizes:
      HEAPP(ON,32,,80,,192,,520,,1232,,2048,)

   . . .
   ```
3. Take the heap values from the ″Suggested Cell Sizes″ line in the storage utilization report and use them in another RPTSTG(ON) function to get another report on storage utilization as shown below:

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON),HEAPPOOLS(ON,32,,80,,192,,520,,1232,,2048,)'
   or
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON),HEAPP(ON,32,,80,,192,,520,,1232,,2048,)'
   ```

   The following example shows the servant joblog output from the RPTOPTS(ON),RPTSTG(ON,32,,80,,192,,520,,1232,,2048,) option.

Example:

```
   .   .
0    HEAP statistics:
       Initial size:                                  83886080

       Increment size:                                 5242880
       Total heap storage used (sugg. initial size):  195803218

       Successful Get Heap requests:                    426551
       Successful Free Heap requests:                   424262
       Number of segments allocated:                         1
       Number of segments freed:                             0
   .   .   .

  Suggested Percentages for current Cell Sizes:
    HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)

  Suggested Cell Sizes:
    HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
 . . .
```

4. Take the heap values from the ″Suggested Percentages for current Cell Sizes″ line of the second storage utilization report and use them in another RPTSTG(ON) function to get a third report on storage utilization as shown below:

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)'
   ```

   The following example shows the servant joblog output from the RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20) option.

   Example:

   ```
      .   .
   0    HEAP statistics:
          Initial size:                                  83886080

          Increment size:                                 5242880
          Total heap storage used (sugg. initial size):  198372130

          Successful Get Heap requests:                    426551
          Successful Free Heap requests:                   424262
          Number of segments allocated:                         1
          Number of segments freed:                             0
      .   .   .

     Suggested Percentages for current Cell Sizes:
       HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)
     Suggested Cell Sizes:
       HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
    . . .
   ```

5. On the third storage utilization report, look for the ″Total heap storage used (sugg. initial size):″ line and use this value for your initial LE heap setting. For example, in the report in third report example this value is 198372130.
6. Make sure that you remove RPTSTG since it does incur a small performance penalty to collect the storage use information.
7. For your client programs that run on z/OS or OS/390®, we recommend that you at least specify HEAPP(ON) on the proc of your client to get the default LE heappools. LE will be providing additional pools (more than 6) and larger than 2048MB cell size in future releases of z/OS. You may be able to take advantage of these increased pools and cell sizes, if you have that service on your system.
8. If you use LE HEAPCHECK, make sure to turn it off once you have ensured that your code doesn't include any uninitialized storage. HEAPCHECK can be very expensive.

Chapter 11. Tuning the WebSphere Application Server for z/OS runtime

To optimize performance, review the following steps involved in tuning the WebSphere Application Server for z/OS runtime.

- "Review the WebSphere Application Server for z/OS configuration"
- "Internal tracing tips for WebSphere for z/OS"
- "Location of executable programs tips for z/OS" on page 66
- "Security tuning tips" on page 103

Review the WebSphere Application Server for z/OS configuration

WebSphere Application Server provides tunable settings for its major components to enable you to make adjustments to better match the runtime environment to the characteristics of your application.

Before you begin

Many applications can run successfully without any changes to the default values for these tuning parameters. Other applications might need changes, for example, a larger heap size, to achieve optimal performance.

About this task

Before you read a description of WebSphere Application Server for z/OS tuning guidelines, it is important to note that, no matter how well the middleware is tuned, it cannot make up for poorly designed and coded applications. Focusing on the application code can help improve performance. Often, poorly written or designed application code changes will make the most dramatic improvements to overall performance.

1. Review the WebSphere for z/OS configuration. One simple way to do this is to look in your application control and server regions in SDSF.
2. When each server starts, the runtime prints out the current configuration data in the joblog.

Internal tracing tips for WebSphere for z/OS

WebSphere Application Server traces can be extremely helpful in detecting and diagnosing problems.

By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.

- Ensure that you are not collecting more diagnostic data than you need.

  You should check your WebSphere for z/OS tracing options to ensure that ras_trace_defaultTracingLevel=0 or 1, and that ras_trace_basic and ras_trace_detail are not set.

  How to view or set: Use the WebSphere administrative console:
  1. Click **Environment > Manage WebSphere Variables**.
  2. On the Configuration Tab check for any of these variables in the name field and observe the variable setting in the value field.
  3. To change or set a variable, specify the variable in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.
- For the best performance with any level of tracing, including ras_trace_defaultTracingLevel=1, set ras_trace_outputLocation to BUFFER. This trace setting stores the trace data in memory, which is later asynchronously written to a CTRACE data set. Setting ras_trace_outputLocation to SYSPRINT or TRCFILE provides approximately the same level of performance, but significantly less than BUFFER.

- You can use the ras_trace_BufferCount and ras_trace_BufferSize settings to control the amount of storage used for trace buffers. Generally, the larger the buffer allocation, the better the performance. However, specifying a buffer allocation that is too large can cause a decrease in performance due to system paging.

  The default settings of ras_trace_BufferCount=4 and ras_trace_BufferSize=1M should perform sufficiently for most applications.
- Make sure you disable JRAS tracing.

  To do this, look for the following lines in the trace.dat file pointed to by the JVM properties file:

  `com.ibm.ejs.*=all=disable`

  `com.ibm.ws390.orb=all=disable`

  Ensure that both lines are set to **=disable** or delete the two lines altogether.

  Note: If ras_trace_outputLocation is set, you may be tracing and not know it.

Location of executable programs tips for z/OS

If you choose to not put most of the runtime in LPA, you might find that your processor storage gets a bigger workout as the load increases.

IBM recommends that you install as much of the WebSphere Application Server for z/OS code in LPA as is reasonable. Also, ensure that you have eliminated any unnecessary STEPLIBs which can affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD`s in the controller and servant procs do not point to any unnecessary libraries. Refer to UNIX System Services (USS) tuning tips for z/OS for USS shared file system tuning considerations.

At a minimum, WebSphere for z/OS will start three address spaces, so that any code that is not shared will load three copies rather than one. As the load increases, many more servants may start and will contribute additional load on processor storage.

Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

Chapter 12. Tuning storage

Running application servers on your z/OS system often requires a high amount of virtual storage. Because virtual storage uses real storage as backup, real storage usage might also be high. Therefore make sure that you do not underestimate the amount of virtual storage that you allocate to running your application servers.

Before you begin

Determine your application server virtual storage requirements based on the number of application servers you are running and the number of requests that each of these servers handles.

About this task

Perform one or more of the following steps if you need to improve client request throughput.

1. Allocate additional virtual storage. The setting of REGION on the JCL for the proc controls the amount of virtual storage available to a z/OS address space. The default values for the WebSphere Application Server controller and servant are set to zero, which tells the operating system to allocate all the available region (close to 2GB). You can limit the amount of virtual storage allocated by setting the REGION parameter to a value other than zero. The size of the JVM heap is the most important factor when determining the setting of the REGION parameter. You should only need to set the REGION to something other than zero when the JVM heap size is very large. The z/OS operating system allocates user storage from the bottom of the address space, which is where the JVM heap is allocated, and system storage from the top. System abends can occur when the system tries to obtain virtual storage and none is available. A non-zero REGION parameter setting prevents this from occurring by preserving storage at the top of the address space for system use. In almost all cases running with the default REGION will be satisfactory.

   Note: For more information on REGION=0M and IEFUSI, please see Installing your application serving environment ection of the information center.
2. Convert application servers with high virtual storage usage to run in 64-bit mode. Running an application server in 64-bit mode allows you to specify larger JVM heap sizes.
3. Convert deployment mangers, that are managing cells in which very large applications are deployed, to run in 64-bit mode.
4. Allocate additional real storage. The total amount of real storage that your system requires depends on the number of servers you are running and the size of the JVM heaps for each server. You should allocate at least 512MB of real storage for a small configuration.

   Recommendation: Sometimes in a heavy use environment, 2GB of central storage is not enough to handle the real storage demands of a high volume Java application. In this situation, you might want to configure your servers to run in 64-bit mode. Running your servers in 64-bit mode gives you the ability to dedicate more central storage to the LPAR, and the ability to define more than 2GB of central storage. When you configure your servers to run in 64-bit mode, all of the storage is defined as central storage.

   The z/OS operating system running on a zSeries processor always runs in 64-bit mode. If you are using a non-zSeries processors, or are running your servers in 31-bit mode, you can minimize paging by defining more expanded storage.

Chapter 13. Workload management (WLM) tuning tips for z/OS

If you are running z/OS Version 1.2 or higher, you can use the administrative console to provide the job control language (JCL) PROC name for the servant and the JCL Parm for the servant and thereby set up a dynamic application environment. Even if you set up a dynamic application environment, you must set the WLM goals for your environment.

Proper WLM goals can significantly affect your application throughput. The WebSphere Application Server address spaces should be given a fairly high priority. When setting the WLM goals for your z/OS system, you might want to:

- Classify location service daemons and controllers as SYSSTC or high velocity.
- Use STC classification rules to classify velocity goals for application servers.

  Java garbage collection runs under this classification. Java garbage collection is a CPU and storage intensive process. If you set the velocity goal too high garbage collection can consume more of your system resources than desired. If your Java heap is correctly tuned, garbage collection for each servant should run no more than 5% of the time. Also, providing proper priority to garbage collection processing is necessary since other work in the servant is stopped during much of the time that garbage collection is running.

  JavaServer Page file compiles run under this classification. If your system is configured to do these compiles at runtime, setting the velocity goal too low can cause longer delays waiting for JavaServer Page file compiles to complete.

  Application work is classified under the work manager.
- Set up an application environment for work running under servants where:
  - The subsystem type is set to CB.
  - The environment is classified based on server name, server instance name, user ID, and transaction class.
  - Percentage response time goals are set.

    You should make the response time goals achievable. For example, a goal that 80% of the work will complete in .25 seconds is a typical goal. Velocity goals for application work are not meaningful and should be avoided.
  - A high velocity default service class for CB subsystem transactions is provided. The default is SYSOTHER.

    Your goals can be multi-period. This might be useful if you have distinctly short and long running transactions in the same service class. On the other hand, it is usually better to filter this work into a different service class if you can. Being in a different service class will place the work in a different servant which allows WLM much more latitude in managing the goals.
- Define unique WLM report classes for servant regions and for applications running in your application environment. Defining unique WLM report classes enables the resource measurement facility (RMF) to report performance information with more granularity.
- Set your Application environment to **No Limit**.
  - Required if you need more than one servant per application server.
  - Under WLM, you can control how many servants can be started for each server. If you need more than one servant in a server make sure that **No Limit** is selected for the application environment associated with your server. For information about setting up WLM performance goals, see *z/OS MVS Planning: Workload Management*.

    Example:

    ```
    Application-Environment  Notes  Options  Help
    ------------------------------------------------------------------------
                       Modify an Application Environment
    Command ===> ____________________________________________________________

    Application Environment Name . : BBOASR2
    Description  . . . . . . . . . . WAS.V40.WB02 Application server
    ```

```
Subsystem Type . . . . . . . . . . CB
Procedure Name . . . . . . . . . . BBOASR2S
Start Parameters . . . . . . . . . IWMSSNM=&IWMSSNM
                                  ____________________________________________
                                  ____________________________________________

Limit on starting server address spaces for a subsystem instance:
1   1.  No limit

    2.  Single address space per system
    3.  Single address space per sysplex
```

When the WLM configuration is set to no limit, you can use the `wlm_maximumSRCount=x` and `wlm_minimumSRCount=y` variables to control the maximum and minimum number of servants. To specify values for these variables, in the administrative console, click **Severs** > **Application servers** and select the appropriate application server.

Note: If you specify a value for the wlm_maximumSRCount variable, the value must be greater than or equal to the number of service classes defined for this application environment. If the value is less than the number of defined service classes, timeouts might be caused because there is an insufficient number of servants available.

– Results reported in RMF Postprocessor workload activity report:
  - Transactions per second (not always the same as client tran rate)
  - Average response times (and distribution of response times)
  - CPU time used
  - Percent response time associated with various delays

Chapter 14. 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 • Replication domains **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 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

Topology planning and performance Topology can have a significant effect on WebSphere performance. This article describes some of the topology considerations you should be aware of when configuring and installing WebSphere Application Server for z/OS.

- Single server or multiple servers?

  WebSphere for WebSphere Application Server for z/OS gives you the ability to install your application either in a single server or spread it across multiple servers. There are many reasons for partitioning your application. However, for performance, placing your application all in the same server will always provide better performance than partitioning it. If you do choose to partition your application across servers, you will get better performance if there are at least replica servers on each system in the sysplex. The WebSphere for WebSphere Application Server for z/OS runtime will try to keep calls local to the system if it can, which will, for example, use local interprocess calls rather than sockets.

- One tran or multiple trans?

  You also have a choice of running server regions with an isolation policy of one tran per server region or multiple trans per server region. From a performance perspective, running more threads in a server region will consume less memory but at the cost of thread contention. This contention is application-dependent. We generally recommend the use of multiple trans unless you run into contention problems.

  Specify the threads setting using the *server_region_workload_profile*. The variables include:
  - ISOLATE - sets the value to 1 thread.
  - CPUBOUND
  - IOBOUND - default
  - LONGWAIT - 40

  The thread value increases with each variable to the maximum number available with the LONGWAIT setting (40). For more information refer to ORB services advanced settings on the z/OS platform

  Note: Please see the ″Servers″ section in the WebSphere Application Server for z/OS information center, access to which can be obtained through the WebSphere Application Server library Web site (http://www-306.ibm.com/software/webservers/appserv/was/library/index.html) for more information on ORB services advanced settings.

- Local client or remote client?

  On a local client, the client and the optimized communication are done on the same system. This has some additional client CPU costs but less communication cost. On a remote client, the client cost is replaced by the additional communication overhead of sockets. The CPU cost on either system is almost equivalent. Latency is better for a local client than for a remote client, meaning you will get better response time with a local client.

- One copy of a server or many clones?

  You can define more than one copy of a server on a system. These copies are called clones. We have found slight improvements in performance when running with a couple of clones as opposed to just one (very large configuration). While there is some benefit, IBM does not recommend, at this time, the creation of replicated control regions for the sole purpose of improving performance. We do, however, recommend them for eliminating a single point of failure and for handling rolling upgrades without introducing an outage.

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 for SOAP
Learn how to tune the soap messages that you use with your Web services.

About this task

SOAP is a lightweight protocol which provides a mechanism to use XML for exchanging structured and typed information between peers in a decentralized, distributed environment.

- Specify noLocalCopies in servant.jvm.options (-Dcom.ibm.CORBA.iiop.noLocalCopies=1). This will enable passing of parameters by reference instead of by value. This only applies if you are exposing an enterprise bean as a web service. Refer to Object Request Broker service settings.

- Make certain that all traces are disabled unless you are actively debugging a problem.
- When defining transaction policies for your application, specify TX_NOT_SUPPORTED and select local transactions. Local transactions perform better than global transactions because WebSphere is not required to coordinate commit scope over multiple resource managers.
- Avoid passing empty attributes or empty elements in SOAP messages. Do not include extraneous and unneeded data in SOAP messages. If you can use document/literal style web services invocation to batch requests into a single SOAP message, this is preferable to sending multiple individual SOAP messages. SOAP applications will perform better with smaller SOAP messages containing fewer XML elements and especially fewer XML attributes. The contents of SOAP messages must be serialized and parsed. These are expensive operations and should be minimized. In other words, it is preferable to send 1, 10KB message than 10, 1KB messages. However, very large messages (for example, over 200KB) might impact system resources like memory.
- You might need to increase the default Java heap size. SOAP and XML (DOM) are storage-intensive and small heap sizes can result in excessive Java garbage collection. We found a heapsize of 256M (the default) was optimal for most test cases in the laboratory. You can monitor garbage collection using the verbose:gc Java directive.
- Insure TCP/IP send/receive buffers are large enough to hold the bulk of the xml messages that will be sent.
- Consider using a Document Model rather than the RPC model. It provides complete control over the format of the XML but requires additional programming effort.
- When using RPC-style messages, try to send strings if possible.
- For Java API for XML-based RPC (JAX-RPC) Web services, consider writing your own serializers and deserializers, avoiding reflection.

SOAP Version 2.3 in WebSphere Application Server:

Learn about the enhancements that SOAP Version 2.3 provides to WebSphere Application Server.

SOAP Version 2.3 has a number of enhancements over SOAP Version 2.2. They are described in full in http://ws.apache.org/soap/releases.html#v2.2. Some of the key enhancements are as follows:
- Support for document-oriented messaging
- Web Services Description Language (WSDL) Version 1.1
- Universal Description, Discovery, and Integration, UDDI4J Version 2.0, logging
- Web Services Invocation Framework (WSIF)
- Web Services Caching
- Some minor SOAP performance enhancements
- Elimination of the Nagle Algorithm

Web Services is a more expensive protocol than HTTP or socket communication. It is best used where its benefits can be exploited. For example, the integration of decentralized distributed environment. We do not recommend that programs running in the same JVM use Web Services as a means of communication or invocation. For obvious reasons, calling a method using SOAP generally has longer response time than other forms of invocation. XML parsing serialization/deserialization and network latency are all inhibitors to Web Services performance. There is no support for locally optimized invocation such that network protocols can be avoided when client and server are collocated.

We have observed, one of the most expensive operations in the processing of a SOAP message, whether by SOAP, AXIS or the Tech preview, is the deserialization of the inbound SOAP message. This step converts the message from an inbound string in wire format to an XML document. Therefore, if either the client or the server receives a large SOAP message, that entity normally has the highest CPU cost. We have found the CPU time to be similar for z/OS acting as either a SOAP server or a SOAP client as long as the inbound and outbound message sizes are comparable.

There are many forms of an XML message that are equivalent, for example, messages with additional white space are equivalent to messages with fewer blanks or spaces. Therefore, it is advisable to create SOAP messages without formatted nesting (pretty printing), which adds additional spaces. The XML parser must examine all characters, including blanks. Therefore, XML documents with additional blank characters will take longer to parse.

Document-oriented messaging, unlike RPC messaging, is not required to be synchronous. The UDDI Registry is a special purpose data base wherein, establishments can register the WSDL Service Interface Definition and the Service Implementation Definition for Web Services. WSDL describes the interfaces to the web service. This essentially makes them available to other establishments or business partners. WebSphere V 5.0 for z/OS also supports a Private UDDI Registry for private access to Web Services.

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**.

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.

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.

      Note: You must set this value if the messaging engine can failover in a cluster.

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.

On z/OS, the performance of messaging applications is affected by the number of servants, which can vary dynamically, and the distribution of work between servants. For more information about configuring and managing the number of servants and the distribution of work between servants, see Tuning the application serving environment.

- 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.

- 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 80
- "Controlling the memory buffers used by a messaging engine" on page 78

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.

      Note: You must set this value if the messaging engine can failover in a cluster.

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.

If you migrate Web services from the WebSphere Application Server Version 5.1 Web services gateway, then your messages use this fast-path route through the bus.

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.

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.
- 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`.

Messaging resources

Tuning JMS destinations

Use this task to configure the properties of a JMS destination to optimize performance of applications that use the WebSphere Application Version 5 default messaging provider or WebSphere MQ as a JMS provider.

About this task

To optimize performance, configure destination properties to best fit your applications. You should also consider queue attributes of the JMS server that are associated with the queue name. For more information, see the following topics:

- "Performance considerations for WebSphere Application Server Version 5 queue destinations"
- "Performance considerations for WebSphere Application Server Version 5 topic destinations" on page 97
- "Performance considerations for WebSphere MQ queue destinations" on page 97
- "Performance considerations for WebSphere MQ topic destinations" on page 98

Performance considerations for WebSphere Application Server Version 5 queue destinations

To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

For queue destinations configured on a WebSphere Application Server Version 5 node, you should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

BOQNAME
: The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

BOTHRESH
: The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

For more information about using these properties, see:

- “Handling poison messages” in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Performance considerations for WebSphere Application Server Version 5 topic destinations

To optimize performance, configure the JMS destination properties to best fit your applications.

For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

For JMS destinations configured on a WebSphere Application Server Version 5 node, ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Performance considerations for WebSphere MQ queue destinations

To optimize performance, configure the queue destination properties to best fit your message-driven beans or other applications that use the queue destinations.

For example:

- When MDB applications are configured to WebSphere MQ queues on z/OS, the INDEX by MSGID is very important.
- Setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for use by your message-driven beans or other applications that use the queue.

You should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

You should also consider the queue attributes associated with the queue name you created with WebSphere MQ. Inappropriate queue attributes can reduce the performance of WebSphere operations. You can use WebSphere MQ commands to change queue attributes for the queue name.

BOQNAME
: The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

BOTHRESH
: The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

INDXTYPE
Set this to MSGID. This causes an index of message identifiers to be maintained, which can improve WebSphere MQ retrieval of messages.

DEFSOPT
Set this to SHARED (for shared input from the queue).

SHARE
This must be specified (so that multiple applications can get messages from this queue).

For more information about using these properties, see:

- For BOQNAME and BOTHRESH, see “Handling poison messages” in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Performance considerations for WebSphere MQ topic destinations

To optimize performance, configure the topic destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

Security

Secure Sockets Layer (SSL) considerations for WebSphere Application Server administrators

The Resource Access Control Facility (RACF) customization jobs create an SSL keyring owned by the WebSphere Application Server for z/OS administrator. This SSL keyring contains the digital certificate needed to communicate with WebSphere Application Server. Other MVS™ user IDs, which require WebSphere Application Server for z/OS administration require additional customization.

Before you begin

The Resource Access Control Facility (RACF) customization jobs create an SSL keyring owned by the WebSphere Application Server for z/OS administrator containing the digital certificate needed to communicate with WebSphere Application Server. However, additional customization is required for administration by other MVS user IDs.

Note that the MVS user ID in the description below is the MVS user ID under which the **wsadmin** process is running, not the user ID specified in the wsadmin request.

About this task

In the example below:

- yyyyy is the user ID of the new WebSphere Application Server for z/OS administrator
- xxxxx is the name of the keyring that is specified in soap.client.props in the *profile_root*/properties directory.
- zzzzz is the label name used in the BBOSBRAK jobs to specify which certificate authority certificate was used to generate server keys

1. If the new administrator is not a member of the WebSphere Application Server for z/OS administrative group, make sure that the new user ID has access to the appropriate RACF keyrings and digital certificates. For example:

   ```
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(yyyyy) ACC(READ)
   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(yyyyy) ACC(READ)
   ```

2. Use the setup completed by the customization jobs as a model for the additional steps. This information is in the BBOCBRAK member of the <HLQ>.DATA data set generated during the customization process. The BBOCBRAK job contains the set of RACF commands that were used:

   ```
     /* Generating SSL keyrings for WebSphere administrator                    */
   RACDCERT ADDRING(xxxxx) ID( yyyyyy )

    /* Connect WebSphere Application Server CA Certificates to Servers keyring    */
   "RACDCERT ID(yyyyy) CONNECT (RING(xxxxx) LABEL('zzzzz')   CERTAUTH"

   SETROPTS RACLIST(FACILITY) REFRESH"
   ```

Setting up SSL connections for Java clients

Follow these steps to configure SSL for use between Java clients running on a workstation and the WebSphere Application Server for z/OS Java Platform, Enterprise Edition (Java EE) server.

1. Determine what SSL repertoire the server is using. For example: WASKeyring.
2. Determine the user ID the server is running. For example: CBSYMSR1.
3. Export the certificate authority from RACF. For example:

   ```
   RACDCERT CERTAUTH EXPORT(LABEL('WebSphereCA')) DSN('IBMUSER.WAS.CA') FORMAT(CERTDER)
   ```

4. Move the file to the workstation. (Note that the FTP transfer must use binary.) For example: \tmp directory
5. Add the digital certificate to the TrustStore used by the client. For example, for the DummyClientTrustFile.jks file, type:

   ```
   keytool -import -file \tmp\IBMUSER.WAS.CA -keystore DummyClientTrustFile.jks]
   ```

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” on page 100.
- 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 107.

- 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 108.

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 propagating new security settings to all nodes before restarting the deployment manager and node agents, to change the new security policy.

    If your security configurations are not consistent across all servers, you get access denied errors. Therefore, you must propagate new security settings when enabling or disabling administrative security.

    Configuration changes are generally propagated using configuration synchronization. If auto-synchronization is enabled, you can wait for the automatic synchronization interval to pass, or you can force synchronization before the synchronization interval expires. If you are using manual synchronization, you must synchronize all the nodes.

    If the cell is in a configuration state and the security policy is mixed with nodes that have security enabled and disabled, you can use the syncNode utility to synchronize the nodes where the new settings are not propagated.

    For more detailed information about enabling security in a distributed environment, see Enabling security for the realm.
  - 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.
  - 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.

- If using SSL, enable the SSL session tracking mechanism option as described in the article, Session management settings.
- 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.

Security tuning tips:

As a general rule, two things happen when you increase security: the cost per transaction increases and throughput decreases. Consider the following security information when you configure WebSphere Application Server.

SAF class

When a SAF (RACF or equivalent) class is active, the number of profiles in a class will affect the overall performance of the check. Placing these profiles in a (RACLISTed) memory table will improve the performance of the access checks. Audit controls on access checks also affect performance. Usually, you audit failures and not successes. Audit events are logged to DASD and will increase the overhead of the access check. Because all of the security authorization checks are done with SAF (RACF or equivalent), you can choose to enable and disable SAF classes to control security. A disabled class will cost a negligible amount of overhead.

EJBROLEs on methods

Use a minimum number of EJBROLEs on methods. If you are using EJBROLEs, specifying more roles on a method will lead to more access checks that need to be executed and a slower overall method dispatch. If you are not using EJBROLEs, do not activate the class.

Java 2 Security

If you do not need Java 2 security, disable it. For instructions on how to disable Java 2 security, refer to Protecting system resources and APIs (Java 2 security).

Level of authorization

Use the lowest level of authorization consistent with your security needs. You have the following options when dealing with authentication:

- **Local authentication:** Local authentication is the fastest type because it is highly optimized.
- **UserID and password authentication:** Authentication that utilizes a userID and password has a high first-call cost and a lower cost with each subsequent call.
- **Kerberos security authentication:** We have not adequately characterized the cost of kerberos security yet.
- **SSL security authentication:** SSL security is notorious in the industry for its performance overhead. Luckily, there is a lot of assists available from hardware to make this reasonable on z/OS.

Level of encryption with SSL

If using Secure Sockets Layer (SSL), select the lowest level of encryption consistent with your security requirements. WebSphere Application Server enables you to select which cipher suites you use. The cipher suites dictate the encryption strength of the connection. The higher the encryption strength, the greater the impact on performance.

RACF tuning

Follow these guidelines for RACF tuning:

- Use the RACLIST to place into memory those items that can improve performance. Specifically, ensure that you RACLIST (if used):
  - CBIND
  - EJBROLE
  - SERVER
  - STARTED

  Example:

  ```
  RACLIST (CBIND, EJBROLE, SERVER, STARTED, FACILITY, SURROGAT)
  ```
- Use of things like SSL come at a price. If you are a heavy SSL user, ensure that you have appropriate hardware, such as PCI crypto cards, to speed up the handshake process.
- Here's how you define the BPX.SAFFASTPATH facility class profile. This profile allows you to bypass SAF calls which can be used to audit successful shared file system accesses.
  - Define the facility class profile to RACF.

    ```
    RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE)
    ```
  - Activate this change by doing one of the following:
    - re-IPL
    - invoke the SETOMVS or SET OMVS operator commands.

  Note: Do not use this option if you need to audit successful HFS accesses or if you use the IRRSXT00 exit to control HFS access.
- Use VLF caching of the UIDs and GIDs as shown in the example COFVLFxx parmlib member below:

  Example: sys1.parmlib(COFVLFxx):

  ```
  ******************************** Top of Data ********************.
  .

  CLASS NAME(IRRGMAP) EMAJ(GMAP)
  CLASS NAME(IRRUMAP) EMAJ(UMAP)
  CLASS NAME(IRRGTS) EMAJ(GTS)
  CLASS NAME(IRRACEE) EMAJ(ACEE)
  .
  ******************************* Bottom of Data ******************
  ```

  To avoid a costly scan of the RACF databases, make sure all HFS files have valid GIDs and UIDs.

Related tasks

Protecting system resources and APIs (Java 2 security)
Java 2 security is a programming model that is very pervasive and has a huge impact on application development.

Related information

Session management settings
Use this page to manage HTTP session support. This support includes specifying a session tracking mechanism, setting maximum in-memory session count, controlling overflow, and configuring session timeout.

Resource Access Control Facility Tips for customizing WebSphere Application Server:

It is important to understand the security mechanisms used to protect the server resources using the CBIND, SERVER, and STARTED classes in RACF (or your security product). This paper describes these mechanisms along with some techniques for managing the security environment.

The first part of this article provides details about the RACF profiles used to protect the WebSphere servers and resources using the following classes:

- CBIND: Access to servers, and access to objects in the servers
- SERVER: Access to controller regions by servant regions
- STARTED: Associate user-ids and groups to started procedures (STCs)

The next part of the article describes adding the required RACF profiles and permissions for another server in your cell.

The last part of the article shows how you can define the minimal set of users, groups, and profiles for a testing environment (where security of individual servers is not the main focus or concern).

RACF Profiles (CBIND, SERVER, and STARTED): Basic information about the RACF profiles used by WebSphere can be found in the System Authorization Facility classes and profiles. This section adds some additional details about the CBIND, SERVER, and STARTED class profiles.

User IDs and Group IDs: As part of using the WebSphere z/OS Profile Management Tool or the **zpmt** command, the BBOCBRAK job generates RACF commands that then can be run with the BBOWBRAK job. Key:

```
CR = Controller Region
SR = Servant Region
CFG = Configuration (group)
server = server short name
cluster = generic server (short) name (also called cluster transition name)
```

First, six users and six groups are defined as follows, which are shown symbolically to help you understand how they are used in the various permissions later on:

```
&lt;CR_userid> &lt;CR_groupid>, &lt;CFG_groupid>
&lt;SR_userid> &lt;SR_groupid>, &lt;CFG_groupid>
&lt;demn_userid> &lt;demn_groupid>, &lt;CFG_groupid>
&lt;admin_userid> &lt;CFG_groupid>
&lt;client_userid> &lt;client_groupid>
&lt;ctracewtr_userid> &lt;ctracewtr_groupid>
```

Below are the various profiles used to protect the WebSphere servers and resources, along with the permissions and access levels.

CBIND Class Profiles: There are two formats and levels of CBIND class profiles for protecting access to application servers and objects in those servers:

```
CBIND Class profiles - access to generic servers
CB.BIND.<cluster> UACC(READ); PERMIT <CR_group> ACC(CONTROL)

CBIND Class profiles - access to objects in servers
CB.<cluster> UACC(READ) PERMIT <CR_group> ACC(CONTROL)
```

SERVER Class Profiles: There are currently two formats of the SERVER class profiles for protecting access to the server controller regions. You must define a single format SERVER profile, depending upon whether or not Dynamic Application Environment (DAE) support is enabled. This is done using the WLM DAE APAR OW54622, which is applicable to z/OS V1R2 or higher.

In the WebSphere z/OS Profile Management Tool or the zpmt command, both formats are predefined, and one of these is actually required at runtime. The required format is determined dynamically by the WebSphere Application Server for z/OS Runtime based on the availability of Dynamic Application Environment (DAE) support.

- The following command provides access to controllers using static Application Environments (without the APAR support): RDEFINE CB.&<server>.&<cluster> UACC(NONE); PERMIT &<SR_userid> ACC(READ) For this example, server = server name, cluster = cluster name or cluster transition name if a cluster has not yet been created, and SR = the MVS user ID of the Server Region.
- The following command provides access to controllers using dynamic Application Environments (with the WLM DAE APAR support): CB.&<server>.&<cluster>.<cell> UACC(NONE); PERMIT &<SR_userid> ACC(READ) For this example, server = server name, cluster = cluster name or cluster transition name if a cluster has not yet been created, cell = cell short name, and SR = the MVS user ID of the Server Region.

STARTED Class Profiles: There are two formats of STARTED class profiles used to assign user and group IDs to controller regions and other STCs based on whether the started task is started with the MGCRE interface or the address space create (ASCRE) interface used by Workload Manager (WLM) to start servant regions:

```
STARTED Class profiles - (MGCRE)
<&lt;CR_proc>.&lt;CR_jobname> STDATA(USER(CR_userid) GROUP(CFG_groupid))
&lt;demn_proc>.* STDATA(USER(demn_userid) GROUP(CFG_groupid))

STARTED Class profiles - (ASCRE)
&lt;SR_jobname>.&lt;SR_jobname> STDATA(USER(SR_userid) GROUP(CFG_groupid))

STARTED Class profiles for IJP - (MGCRE)
&lt;MQ_ssname>.* STDATA(USER(IJP_userid) GROUP(CFG_groupid))
```

Generating new user IDs and Profiles for a new Server: If you want to use unique user IDs for each new application server, you must define these users, groups, and profiles in the RACF database.

One technique is to edit a copy of the BBOWBRAK member using the WebSphere z/OS Profile Management Tool or the zpmt command, .DATA partitioned data set, and change the following entries to the new users, groups, and unique New_server name, and New_cluster name profiles:

- If unique user IDs for the new servers are desired, define three new users and connect them to the following groups:

  ```
  &lt;New_CR_userid> &lt;CR_groupid>, &lt;CFG_groupid>
  &lt;New_SR_userid> &lt;SR_groupid>, &lt;CFG_groupid>
  &lt;New_client_userid> &lt;client_groupid>
  ```

- CBIND class profiles for the new cluster (generic server short name):

  ```
  CB.BIND.&lt;New_cluster>
  CB.&lt;New_cluster>
  ```

- SERVER class profiles for the new server and cluster:

  ```
  CB.&lt;New_server>.&lt;New_cluster>
  CB.&lt;New_server>.&lt;New_cluster>.&lt;cell>
  ```

- STARTED class profiles for the new server's controller and servant's regions:

  ```
  &lt;CR_proc>.&lt;New_CR_jobname> STDATA(USER(New_CR_userid)
                                        GROUP(CFG_groupid))
  &lt;New_SR_jobname>.* STDATA(USER(New_SR_userid) GROUP(CFG_groupid))
  ```

Minimalist Profiles: To minimize the number of users, groups, and profiles in the RACF data set, you can use one user ID, one group ID, and very generic profiles so they cover multiple servers in the same cell. Here is an example of profiles with one user (T5USR), one group (T5GRP), and a set of servers in the T5CELL having server short names starting with T5SRV* and generic server names starting with T5CL*. This technique can also be used with Integral JMS provider (IJP) and Network Deployment (ND) configurations.

```
/* CBIND Class profiles (UACC) - access to generic servers */
CB.BIND.T5CL* UACC(READ); PERMIT ID(T5GRP) ACC(CONTROL)

/* CBIND Class profiles (UACC) - access to objects in servers */
CB.T5CL* UACC(READ); PERMIT ID(T5GRP) ACC(CONTROL)
```

```
/* SERVER Class profiles - access to controllers (old style) */
CB.*.T5CL* UACC(NONE); PERMIT ID(T5USR) ACC(READ)

/* SERVER Class profiles - acc to controllers (new style) */
CB.*.*.T5CELL UACC(NONE); PERMIT ID(T5USR) ACC(READ)

/* STARTED Class profiles - (MGCRE) - for STCs, except servants */
T5ACR.* STDATA(USER(T5USR) GROUP(T5GRP)) /* controller*/
T5DMN.* STDATA(USER(T5USR) GROUP(T5GRP)) /* daemon */
T5CTRW.* STDATA(USER(T5USR) GROUP(T5GRP)) /* CTrace WTR*/
WMQX*.* STDATA(USER(T5USR) GROUP(T5GRP)) /* IJP */

/* STARTED Class profiles - (ASCRE - for servants) */
T5SRV*.* STDATA(USER(T5USR) GROUP(T5GRP)) /* servant */
```

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.

While it is not practical to run WebSphere Application Server for z/OS without security enabled, it is possible to perform certain tuning techniques to make the Application Server run better on z/OS. These techniques are documented in detail in “Security tuning tips” on page 103.

- 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.
- Modify the RACF security settings as documented in the “Security tuning tips” on page 103 article.
- Read the “Tuning security configurations” on page 100 article 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 111.

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 2. 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
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 2. 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 • Session persistence password • DRS client data replication 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
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

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

Table 3. 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	
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 z/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.

The reliance on passwords in configuration files can be minimized on WebSphere Application Server for z/OS by taking advantage of z/OS-specific features:

- Use a System Authorization Facility (SAF) registry to remove the requirement for a user registry server password.
- Select SAF authorization and delegation so role-to-user binding passwords are removed.
- Use a RACF keyring for all SSL repertoires, and trust and key file passwords are no longer required.
- Use native connectors, and configure sync-to-thread to possibly remove the need for Java Authentication and Authorization Service (JAAS) authentication data.

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. The order in which enablement occurs is important. When adding a new node to a cell that contains password encryption, the new node must contain the custom encryption classes prior to using the **addNode** command. Consider the following Network Deployment enablement scenarios:
   a. The StandAloneProfile profile is encrypting passwords with a different key prior to federation to a deployment manager cell. For this scenario, you must uninstall custom password encryption to ensure that the configuration has `{xor}` tags preceding the passwords prior to running the **addNode** command. The same implementation of the plug point must be in the `/classes` directory prior to running the **addNode** command, and the proper configuration properties are set so that the new node can recognize the encrypted password format of the `security.xml` file after federation completes.
   b. The StandAloneProfile profile does not have password encryption configured prior to federation to a deployment manager cell. The same implementation of the plug point must be in the `/classes` directory prior to running the **addNode** command, and the proper configuration properties are set so that the new node can recognize the encrypted password format of the `security.xml` file after federation completes.
   c. If enabling custom password encryption in a cell with multiple nodes present, update the correct configuration properties and have the custom password encryption implementation class located on all nodes. Stop all processes in the cell, and then start the deployment manager. Use the administrative console to edit the security configuration and then save it. Verify that the passwords are encrypted by looking at the `security.xml` file to see if the passwords are preceded by `{custom:alias}` tags.
   d. Run the **syncNode** command on each node, and start each one individually. If any nodes fail to start, make sure that they have custom password encryption enabled properly in each `security.xml` file and that the implementation class is in the appropriate `/classes` directory for the platform.

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.

Cache configuration value	Description
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 15. Troubleshooting performance problems

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

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.

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

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

only use synchronization when necessary; reducing the path length of the synchronized code; or reducing the frequency of invoking the synchronized code.

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 (z/OS)

app_server_root
Refers to the top directory for a WebSphere Application Server node.

The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own *app_server_root*. Corresponding product variables are was.install.root and WAS_HOME.

The default varies based on node type. Common defaults are *configuration_root*/AppServer and *configuration_root*/DeploymentManager.

configuration_root
Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

The *configuration_root* contains the various *app_server_root* directories and certain symbolic links associated with them. Each different node type under the *configuration_root* requires its own cataloged procedures under z/OS.

The default is /wasv7config/*cell_name*/*node_name*.

plug-ins_root
Refers to the installation root directory for Web Server plug-ins.

profile_root
Refers to the home directory for a particular instantiated WebSphere Application Server profile.

Corresponding product variables are server.root and user.install.root.

In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will be always be *app_server_root*/profiles/default because only the profile name ″default″ is used in WebSphere Application Server for z/OS.

smpe_root
Refers to the root directory for product code installed with SMP/E.

The corresponding product variable is smpe.install.root.

The default is /usr/lpp/zWebSphere/V7R0.

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA
Attention: Information Requests

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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.