WebSphere® Application Server for z/OS, Version 6.0.1

IBM

Tuning guide

SA22-7963-03

Note

Before using this information, be sure to read the general information under "Notices" on page 101.

Compilation date: March 1, 2005

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 919-254-0206.

  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.

Chapter 1. Overview and new features for tuning performance

New for administrators: Improved monitoring and performance tuning
A section of this topic describes what is new in the area of performance tuning.

Presentations from IBM Education Assistant
- Performance overview
- Caching, including dynamic cache
- Performance advisors and Tivoli Performance Viewer (TPV)

Contents of this section: Tuning performance

Chapter 2. How do I tune performance?

Legend for ″How do I?...″ links

Documentation	Show me	Tell me	Guide me	Teach me
Refer to the detailed steps and reference	Watch a brief multimedia demonstration	View the presentation for an overview	Be led through the console pages	Perform the tutorial with sample code
Approximate time: Varies	**Approximate time:** 3 to 5 minutes	**Approximate time:** 10 minutes+	**Approximate time:** 1/2 hour+	**Approximate time:** 1 hour+

Tune the application serving environment

Each WebSphere Application Server process has several parameters influencing application performance.

Documentation

Obtain tuning advice

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

Documentation	Show me	Tell me
	• Performance advisor • TPV advisor	

Tune WebSphere applications

Expand **Tuning performance > Tuning WebSphere applications** in the information center table of contents for information about tuning specific application components and their administrative configurations (such as EJB container settings).

Pool database connections

As described in Connection pooling, when accessing any database, establishing connections is an expensive operation. Connection pooling enables administrators to establish a pool of database connections that applications can share on an application server. When connection pooling capabilities are used, performance improvements up to 20 times the normal results are realized.

Documentation

Configure caching

The dynamic cache service improves performance by caching the output of servlets, commands, and JSP files.

Documentation Show me Tell me

Chapter 3. Planning ahead 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

  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 swamped 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.

  A concise guide to WebSphere capacity management

  How much capacity is enough for your Web site? It is a difficult question, and often customers have either too much or too little capacity to meet the demands of their site visitors. When developing a plan for your Web site, you must first define its service levels. Next, you must determine the health of your existing site and apply what you know to planning for your future site. Finally, you must validate your capacity estimates through testing and then eventually deploy the site. Here, developing and implementing a capacity plan for your site is explored.

  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 the implementation of tuning applications.

Consult the ″Designing applications″ topic, in the *Developing and Deploying applications* book, 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 2 Platform, Enterprise Edition (J2EE) specification.

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

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

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

Persistence

Java 2 Platform, Enterprise Edition (J2EE) 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 cleanly architected solution that 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 J2EE 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 in the *Setting up the application serving environment* book 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 in the *Setting up the application serving environment* book.

Model-view-controller pattern

One of the standard J2EE 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:
- "Tuning session management" on page 66
- "EJB Container Cache tuning" on page 75
- "Tuning dynamic cache with the cache monitor" on page 93

Caching

Most J2EE 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 J2EE 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

J2EE 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 J2EE 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) J2EE best practices

Improve performance in your XML applications, Part 2

Chapter 4. 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.

  Task overview: 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 5 (architectural suggestions)

  – The ″Designing applications″ topic in the in the *Developing and Deploying applications* book

Chapter 5. Obtaining performance advice from the performance advisors

Tuning WebSphere Application Server is a critical part of getting the best performance from your Web site. But 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.

Why use the performance advisors?

The performance advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere-specific rules for tuning. The performance 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 performance 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
- Prepared statement cache size
- Session cache size
- Dynamic cache size
- Java virtual machine heap size
- DB2 Performance Configuration Wizard

For example, consider the data source-prepared statement cache, which helps to decrease the cost of running a prepared statement by caching the compiled statement. 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.

Using another example with pools in the application server, 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.

Choosing the right performance advisor for the purpose

Two performance advisors are available: the Runtime Performance Advisor and the performance advisor in Tivoli Performance Viewer (TPV advisor). The Runtime Performance Advisor runs in the Java virtual machine (JVM) process of Application Server; therefore, it does not provide expensive advice. In a Network Deployment environment, the performance advisor in TPV runs on the node agent and can provide advice on resources which are more expensive to monitor and analyze. The TPV advisor requires that you enable performance modules, counters, or both.

The following chart shows the differences between the Runtime Performance Advisor and the TPV advisor:

	Runtime Performance Advisor	**TPV advisor**
Start location	Application server	TPV client
Invocation of tool	TPV	Administrative console
Output	The SystemOut.log file and the administrative console	TPV in the administrative console
Frequency of operation	Configurable	When you select refresh in the TPV administrative console
Types of 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	• 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

Runtime Performance Advisor

The Runtime Performance 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 Runtime Performance 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 WebSphere Runtime Messages in the WebSphere Status panel and as text in the application server `SystemOut.log` file. Enabling the Runtime Performance Advisor has minimal system performance impact.

The advice the Runtime Performance 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 Runtime Performance Advisor gives two sets of advice (one for `server1` and another for `server2`). You must be aware that 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 declaring two different data sources for each server with the appropriate level.

Tivoli Performance Viewer advisor

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.

Using the Runtime Performance Advisor

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

1. Enable PMI in the application server administrative console as described in the *Administering applications and their environment* book. To obtain advice, you must first enable PMI through the administrative console and restart the server. The Runtime Performance Advisor enables the appropriate monitoring counter levels for all enabled advice when PMI is enabled. If specific counters exist that are not wanted, disable the corresponding advice in the Runtime Performance Advisor panel, and disable unwanted counters. If specific counters exist that are not wanted, or when disabling the Runtime Performance Advisor, you might want to disable PMI or the counters that the Runtime Performance Advisor enabled.
2. Click **Servers** > **Application servers** in the console navigation tree.
3. Click *server_name* > **Runtime Performance Advisor Configuration**.
4. Under the Configuration tab, specify the number of processors on the server. This setting is critical to ensure accurate advice for the specific configuration of the system.
5. Select the **Calculation Interval**. PMI data is taken over an interval of time and averaged to provide advice. The calculation interval specifies the length of the time over which data is taken for this advice. Therefore, details within the advice messages display as averages over this interval.
6. Select the **Maximum Warning Sequence**. The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is updated. For example, if the maximum warning sequence is set to 3, then the advisor 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.

7. Click **Apply**.
8. Click **Save**.
9. Click the **Runtime** tab.
10. Click **Restart**. Select **Restart** on the Runtime tab to re-initialize the Runtime Performance Advisor using the last configuration information that is saved to disk.

    Note: This action also resets the state of the Runtime Performance Advisor. For example, the current warning count is reset to zero (0) for each message.
11. Simulate a production level load. If you use the Runtime Performance 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 Runtime Performance Advisor only provides advice when CPU utilization exceeds a sufficiently high level. For a list of IBM business partners providing tools to drive this type of load, see the "Performance: Resources for learning" topic in the *Administering applications and their environment* book.
12. Select the check box to enable the Runtime Performance Advisor.

    Note: To achieve the best results for performance tuning, enable the Runtime Performance Advisor when a stable production level load is being applied.
13. Click **OK**.
14. Select **Warnings** in the administrative console under the WebSphere Runtime Messages in the WebSphere Status panel or look in the `SystemOut.log` file, which is located in the *`install_root`*`/profiles/`*`profile_name`*`/logs/`*`server_name`* directory to view tuning advice. Some messages are not issued immediately.
15. 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 pervious configuration.

    Over time, the advisor might issue differing advice. The differing advice is due to load fluctuations and the run-time state. When differing advice is received, you need to look at all advice and the time period over which it ia 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.

You can enable and disable advice in the Advice Configuration panel. Some advice applies only to certain configurations, and can only be enabled 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 article, "Advice configuration settings" on page 16.

Runtime Performance Advisor configuration settings

Use this page to specify settings for the Runtime Performance Advisor.

For more information on how to use the Runtime Performance Advisor, see the article, "Using the Runtime Performance Advisor" on page 13.

To view this administrative page, click **Servers** > **Application Servers** > *server_name* > **Runtime Performance Advisor Configuration** under the Performance section.

Enable Runtime Performance Advisor

Specifies whether the Runtime Performance Advisor runs on the server startup.

The Runtime Performance 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 Runtime Performance Advisor is not enabled, the Runtime Performance Advisor enables it automatically. When disabling the Runtime Performance Advisor, you might want to disable Performance Monitoring Infrastructure (PMI) or the counters that Runtime Performance Advisor enabled. The following counters might be enabled by the Runtime Performance 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

Calculation Interval

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.

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.

Number of processors

Specifies the number of processors on the server.

This setting is critical to ensure accurate advice for the specific configuration of the system.

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* > **Runtime Performance Advisor Configuration** under the Performance section > **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 run-time performance advice applies.

Advice status

Specifies whether the advice is stopped or started.

Specify one of two values: **Started** and **Stopped**. **Started** means that the advice runs if the advice applies. **Stopped** means that the advice does not run.

Advice status

Specifies whether the advice is stopped, started or unavailable.

The advice status has one of three values: **Started**, **Stopped** or **Unavailable**. **Started** means that the advice is applied. **Stopped** means that the advice is not applied. **Unavailable** means that the advice does not apply to the current configuration, for example, persisted session size advice in a configuration without persistent sessions.

Viewing the Runtime Performance Advisor recommendations

The Runtime Performance Advisor recommendations are displayed at the following locations:

- The WebSphere Application Server log file `SystemOut.log`. 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 TUNE0202W:
  Increasing the Web Container thread pool's Maximum Size to 48
  may 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.
```

- The Runtime Messages panel in the administrative console. To view this administrative page, click **Troubleshooting** > **Runtime Messages** > **Warning**.

Related tasks

"Using the Runtime Performance Advisor" on page 13

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. Obtain advice by clicking **Performance advisor** in TPV. The performance advisor in TPV provides more extensive advice than the Runtime Performance Advisor. 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 *Administering applications and their environment* book. 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 node agent before restarting the server and the node 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 "Performance: Resources for learning" topic in the *Administering applications and their environment* book.

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 ″Configure TPV settings″ article in the *Administering applications and their environment* book.
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 6. Tuning the application serving environment

This topic describes the benefits of tuning for optimal performance, highlights the tunable parameters of the major WebSphere Application Server components, and provides insight about how these parameters affect performance.

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

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

Tuning parameter index for z/OS

Performance tuning for WebSphere Application Server for z/OS becomes a complex exercise because the nature of the runtime involves many different components of the operating system and middleware. Use the “Tuning index for WebSphere Application Server for z/OS” on page 20 to find information and parameters for tuning the z/OS operating system, subsystems, the WebSphere Application Server for z/OS run-time environment, and some Java 2 Platform, Enterprise Edition (J2EE) application tuning tips.

Recommendation: 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.

The tuning guide focuses on server tuning. If you want to tune your applications, see the ″Performance: Resources for learning″ article in the *Administering applications and their environment* book.

For your convenience, procedures for tuning parameters in other products, such as DB2, Web servers and operating systems are included. Because these products might change, consider these descriptions as suggestions.

Each WebSphere Application Server process has several parameters that influence application performance. You can use the WebSphere Application Server administrative console to configure and tune applications, Web containers, Enterprise JavaBeans (EJB) containers, application servers and nodes in the administrative domain.

If you are a WebSphere Application Server administrator or Systems programmer on WebSphere Application Server for z/OS, refer to “Tuning index for WebSphere Application Server for z/OS” on page 20 for z/OS specific tuning tips.

Each parameter description: explains the parameter; provides reasons to adjust the parameter; discusses how to view or set the parameter; as well as indicates default and recommended values.

Additional references:

- WebSphere Application Server - Performance Web site

Recommended hardware configuration

The tuning guidelines presented in this chapter will have the most benefit on the following recommended configuration:

- 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
  - 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

  To maximize your performance, we recommend a fast DASD subsystem (for example, IBM Shark), running with a high cache read/write hit rate.
- Networking

  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

  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.

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 four layers of tuning that need to be addressed:

- "Tuning the z/OS operating system"
- "Tuning for subsystems" on page 31
- "Tuning the WebSphere Application Server for z/OS runtime" on page 44
- "Tuning for J2EE applications" on page 46

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 the *Developing and deploying applications* PDF book.

Tuning the z/OS operating system

Steps involved in tuning the z/OS operating system to optimize WebSphere performance include:
- "Tuning storage"
- "z/OS or OS/390 operating system tuning tips" on page 23
- "Resource Recovery Service (RRS) tuning tips for z/OS" on page 24
- "UNIX System Services (USS) tuning tips for z/OS" on page 28
- "Workload management (WLM) tuning tips for z/OS" on page 29

Tuning storage

WebSphere for z/OS puts much higher demands on virtual memory than a traditional workload. Ensure that you don't underestimate the amount of virtual storage applied to the WebSphere for z/OS servers. Generally, they use significantly more virtual memory than traditional application servers on z/OS. Since real storage is needed to back the virtual storage, real storage usage may also be high.

1. Allocate enough virtual storage. The setting of REGION on the JCL for the proc should be large (at least 200MB to run), and much larger if high throughput is required. You can get an idea of the virtual storage usage through RMF or other performance monitors. It would not be unreasonable for the servant procs to specify REGION=0M, which tells the operating system to give all the available region (close to 2GB).

   Note: For more information on REGION=0M and IEFUSI, please see "Preparing the base operating system" in the Getting Started section of the information center.

   If you choose to not put most of the runtime in LPA, as described in the program locations section, be sure to specify more region (as high as 512MB). Also, in conjunction with the increase in storage usage, you may have to define more paging space or auxiliary storage to back up the additional virtual storage used.
2. **Optional:** Allocate enough real storage. Expect a requirement of at least 384MB of real storage for a small configuration. For controllers and servants, real storage utilizations depends on the size of the JVM heapsize.

   Recommendation: It can be the case that in a heavy use environment 2G of central storage is not enough to handle the real storage demands of a high volume Java application. In this case, we recommend that you configure with 64-bit real storage, which will give you the ability to dedicate more central storage to the LPAR. z/OS on a zSeries will always run in 64-bit mode. Running in 64-bit mode gives you the ability to define more than 2 GB of central storage When you configure for 64-bit real all of the storage is defined as central storage. For non-zSeries processors, or 31-bit mode, you can minimize paging by defining more expanded storage.

3. "Java virtual machine storage tuning tips for z/OS." Also refer to the *Troubleshooting and support* PDF book.

Java virtual machine storage tuning tips for z/OS: Specifying a sufficient `JVM Heap Size` is important to Java performance. The JVM has thresholds it uses to manage the JVM's storage. When the thresholds are reached, the garbage collector (GC) gets invoked to free up unused storage. GC can cause significant degradation of Java performance.

Use the administrative console to specify the Initial Heap Size and the Maximum Heap Size for the JVM.

To view this administrative console page, click **Servers > Application Servers** >*server_name* **> Process Definition > Java Virtual Machine**. Access the configuration tab to change these settings.

- In order to get it to run less frequently, you can give the JVM more memory. This is done by specifying a larger value for `Initial Heap Size`. The default of 256M is a good starting point but may need to be raised for larger applications. When specifying either a larger or smaller JVM heap size value, IBM recommends that you code **both** the initial and maximum values that you desire.
- It is good for the Initial Heap Size to equal the Maximum Heap Size because it allows the allocated storage to be completely filled before GC kicks in. Otherwise, GC will run more frequently than necessary, potentially impacting performance.
- Make sure the region is large enough to hold the specified JVM heap.
- Beware of making the `Initial Heap Size` *too* large. While it initially improves performance by delaying garbage collection, it ultimately affects response time when garbage collection eventually kicks in (because it runs for a longer time).
- Paging activity on your system must also be considered when you set your JVM heap size. If your system is already paging heavily, increasing the JVM heap size might make performance worse rather than better.
- To determine if you are being affected by garbage collection, you can enable Verbose Garbage Collection on the JVM Configuration tab. The default is not enabled. This will write a report to the output stream each time the garbage collector runs. This report should give you an idea of what is going on with Java GC.

  Example: This is an example of a verboseGC report.

  ```
  ..
  <AF[21]: Allocation Failure. need 32784 bytes, 32225 ms since last AF
  >
  <AF[21]: managing allocation failure, action=1 (84320/131004928)
                                                 (3145728/3145728)>
  <GC(21): GC cycle started Wed Feb 27 22:46:11 2002
  <GC(21): freed 99587928 bytes, 76% free (102817976/134150656),
   in 118 ms>
  <GC(21): mark: 103 ms, sweep: 15 ms, compact: 0 ms>
  <GC(21): refs: soft 0 (age >= 32), weak 0, final 878, phantom 0>
  <AF[21]: completed in 118 ms
  >.
  ..
  ```

  Key things to look for in a verboseGC report are:
  - Time spent in garbage collection.

    Ideally, you want to be spending less than 5% of the time in GC. To determine percentage of time spent in GC, divide the time it took to complete the collection by the time since the last AF and multiply the result by 100. For example,

```
118/32225 * 100 = 0.366%
```

If you are spending more than 5% of your time in GC and if GC is occurring frequently, you may need to increase your Java heap size.

  – Growth in the allocated heap.

    To determine this, look at the %free. You want to make sure the number is not continuing to decline. If the %free continues to decline you are experiencing a gradual growth in allocated heap from GC to GC which could indicate that your application has a memory leak.

You can also use the MVS console command, *modify display, jvmheap* to display JVM heap information. See "Modify command" for details. In addition, you can check the server activity and interval SMF records. The JVM heap size is also made available to PMI and can be monitored using the Tivoli Performance Viewer.

z/OS or OS/390 operating system tuning tips

This section provides tuning tips for various components of z/OS or OS/390.

- 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):

  ```
  TRACE CT,OFF,COMP=xxxx
  ```

  This will eliminate any unnecessary overhead of collecting trace information that is not being used. 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 SMFPRMxx to ensure that only the minimum number of records are being 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 may find that running with SMF 120 records in production is appropriate, since these records give information specific to WebSphere applications such as response time for J2EE artifiacts, 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. For details of the SMF 120 record, refer to the *Troubleshooting and support* PDF book.

For steps involved in controlling collection of SMF 120 records refer to theEnabling SMF recording topic in the *Troubleshooting and support* PDF book. 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_activity_enabled = true)
- server_SMF_container_activity_enabled=0 (or false)
- server_SMF_container_interval_enabled=1 (or true)
- server_SMF_interval_length=1800

- You might also want to review your DB2 records and the standard RMF written SMF records, and ensure that the SMF datasets are allocated optimally. Information on ensuring a high performance SMF recording environment can be found in the chapter on Customizing SMF in *z/OS MVS System Management Facilities (SMF)*.

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

- For best throughput, use coupling facility (CF) logger.

  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 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)
```

LE tuning tips for z/OS

- 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.

  Processes that run the client ORB, since they start the JVM, must run with xplink(on) For best performance in the xplink runtime, compile applications that use JNI services with xplink enabled. Enabling xplink in the runtime environment and compiling applications with xplink enabled improves performance in z/OS V1R2. As you move from z/OS V1R2 to z/OS V2R6 you should experience additional performance improvements when all the LE services WebSphere calls are xplink enabled. Not all native code in WebSphere Application Server for z/OS or companion products such as DB2 or MQ are xplink enabled and this is an area of active investigation.
- 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"

Fine tuning the LE heap:

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,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.

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

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
  ```

Workload management (WLM) tuning tips for z/OS

If you are running z/OS 1.2 or higher, you might choose to use the dynamic application environment. In that case, use the administrative console to turn on the dynamic application environment. Use the administrative console to provide the job control language (JCL) proc name for the servant and the JCL Parm for the servant. Use the following instructions to set the WLM goals even if you do not use the dynamic application environment.

Setting the WLM goals properly can have a very significant effect on application throughput. The WebSphere for z/OS system address spaces should be given a fairly high priority. As work comes into the system, the work classification of the enclaves should be based on your business goals.

- 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 GC is a CPU and storage intensive process, so if you set the velocity goal too high GC could consume more of the system resources than desired. On the other hand, if your Java heap is correctly tuned, GC for each server region should run no more than 5% of the time. Also, providing proper priority to GC processing is necessary since other work in the server region is stopped during much of the time GC is running.

  JSP compiles run under this classification. If your system is configured to do these compiles at runtime, setting the velocity goal too low could result in longer delays waiting for JSP compiles to complete.

  Application work is actually classified under the work manager.
- Application environment for work running under servants
  - Subsystem type = CB
  - Classify based on server name, server instance name, User ID, and transaction class
  - Percentage response time goal is recommended

    It is usually a good idea to make the 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.
  - Provide a high velocity default service class for CB transactions (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.
- You should define unique WLM report classes for servant regions and for applications running in your application environment. By performing this action, the resource measurement facility (RMF) can 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
```

Note: When the WLM configuration is set to no limit, you can control the maximum and minimum number of servants by specifying the product variables wlm_maximumSRCount=*x* and wlm_minimumSRCount=*y*. To specify these variables, click **Severs** > **Application servers** and go the application server page. **Caution:** If you specify wlm_maximumSRCount you must ensure that you specify a wlm_maximumSRCount value that is greater than or equal to the number of service classes you have defined for this application environment. Failure to do so can result in timeouts due to an insufficient number of servants.

– Results reported in RMF Postprocessor workload activity report:rprf_tunezwlm
  - 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

Tuning for subsystems

Steps involved in tuning the z/OS subsystems to optimize WebSphere performance include:
- DB2 tuning tips for z/OS
- RACF tuning tips for z/OS
- TCP/IP tuning tips for z/OS
- Tuning WebSphere MQ as a JMS provide
- GRS tuning tips for z/OS
- Java virtual machine (JVM) tuning tips for z/OS
- CICS tuning tips for z/OS

DB2 tuning tips for z/OS

Performance tuning for DB2 is usually critical to the overall performance of a WebSphere Application Server application. DB2 is often the preferred datastore for a session or EJB. There are many books that cover DB2 tuning—we can't possibly provide as thorough a treatment of DB2 here as we would like. Listed here are some basic guidelines for DB2 tuning as well as some guidelines for tuning DB2 for WebSphere Application Server. For more complete information on DB2 tuning refer to the DB2 Universal Database for OS/390 and z/OS Administration Guide Document Number SC26-9931-03. The DB2 books can be accessed at the following Internet location:http://www.ibm.com/servers/eserver/zseries/zos/.

You have the choice of Java Database Connectivity (JDBC) or, with WebSphere Application Server Version 5.0.1, Structured Query Language in Java (SQLJ) support with both BMP and CMP beans when you are using the DB2 Universal JDBC driver provider with DB2 as your backend database. JDBC makes use of dynamic SQL whereas SQLJ generally is static and uses pre-prepared plans. You can also use SQLJ support with BMP beans when you are using the DB2 for z/OS Local JDBC provider (RRS) with DB2 for z/OS as your backend database. DB2 for z/OS users who wish to use SQLJ support with CMP beans must use the DB2 Universal JDBC driver provider. SQLJ requires an extra step to create and bind the plan whereas JDBC does not. SQLJ, as a general rule, is faster than JDBC. From a performance point of view, SQLJ it is generally a better performer than JDBC on z/OS DB2. Other features of SQLJ are that it's security model and it's repeatbility are similar to static SQL which is often preferred by z/OS DB2 DBAs. SQLJ does require a couple of extra steps that are features of newer versions of WebSphere Studio Application Developer and Rational Application Developer. Refer to Developing data access applications.

General DB2 tuning tips:

This discussion relates only to DB2 for z/OS JDBC Driver which is referred to as the DB2 for z/OS Legacy JDBC Driver.

- First, ensure that your DB2 logs are large enough, are allocated on the fastest volumes you have, and make sure they have optimal CI sizes.
- Next, ensure that you have tuned your bufferpools so that the most often-read data is in memory as much as possible. Use ESTOR and hyperpools.
- You many want to consider pre-formatting tables that are going to be heavily used. This avoids formatting at runtime.

DB2 for WebSphere tuning tips:

- Ensuring DB2 Tracing Under the DB2 for z/OS Universal Driver is Turned Off.
  - If the db2.jcc.propertiesFile jvm property has been defined to specify a DB2 jcc properties file to the WebSphere Application Server for z/OS, ensure that the following trace statements in the file are commented out if they are specified:

    ```
    # jcc.override.traceFile=<file name>
    # jcc.override.traceFile=<file name>
    ```

  - If any of the DB2 Universal JDBC Driver datasources your applications are using are defined with a nonzero traceLevel custom property, use the WebSphere Application Server for z/OS Administrative console to set the traceLevel to zero.
- We recommend that you ensure indexes are defined on all your object primary keys. Failure to do so will result in costly tablespace scans.
- Ensure that, once your tables are sufficiently populated, you do a re-org to compact the tables. Executing RUNSTATS will ensure that the DB2 catalog statistics about table and column sizes and accesses are most current so that the best access patterns are chosen by the optimizer.
- You will have to define more connections called threads in DB2. WebSphere Application Server uses a lot of threads. Sometimes this is the source of throughput bottlenecks since the server will wait at the create thread until one is available.
- Make sure you are current with JDBC maintenance. Many performance improvements have been made to JDBC. To determine the JDBC maintenance level, enter the following from the shell:

  ```
  java  com.ibm.db2.jcc.DB2Jcc  -version
  ```

  If this returns a class not found, either you are at a level of the driver that is older and doesn't support this command or you have not issued the command properly.
- We recommend that you enable dynamic statement caching in DB2. To do this, modify your ZPARMS to say `CACHEDYN(YES) MAXKEEPD(16K)`. Depending on the application, this can make a very significant improvement in DB2 performance. Specifically, it can help JDBC and LDAP query.
- Increase DB2 checkpoint interval settings to a large value. To do this, modify your ZPARMS to include `CHKFREQ=xxxxx`, where xxxxx is set at a high value

when doing benchmarks. On production systems there are other valid reasons to keep checkpoint frequencies lower, however.

Example: This example identifies zparm values discussed in this article.

```
//DB2INSTE   JOB MSGCLASS=H,CLASS=A,NOTIFY=IBMUSER
/*JOBPARM SYSAFF=*
//******************************************************************
//* JOB NAME = DSNTIJUZ
//*
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5675-DB2
//*    (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 7
//*
//* FUNCTION = DSNZPARM AND DSNHDECP UPDATES
//*
//* PSEUDOCODE =
//*   DSNTIZA  STEP  ASSEMBLE DSN6.... MACROS, CREATE DSNZPARM
//*   DSNTIZL  STEP  LINK EDIT DSNZPARM
//*   DSNTLOG  STEP  UPDATE PASSWORDS
//*   DSNTIZP  STEP  ASSEMBLE DSNHDECP DATA-ONLY LOAD MODULE
//*   DSNTIZQ  STEP  LINK EDIT DSNHDECP LOAD MODULE
//*   DSNTIMQ  STEP  SMP/E PROCESSING FOR DSNHDECP
//*
//* NOTES = STEP DSNTIMQ MUST BE CUSTOMIZED FOR SMP.  SEE THE NOTES
//*         NOTES PRECEDING STEP DSNTIMQ BEFORE RUNNING THIS JOB.
//*
//*  LOGLOAD=16000000,
//*********************************************************************/
//*
//DSNTIZA EXEC PGM=ASMA90,PARM='OBJECT,NODECK'
//STEPLIB DD DSN=ASM.SASMMOD1,DISP=SHR
//SYSLIB   DD  DISP=SHR,
//         DSN=DB2710.SDSNMACS
//         DD  DISP=SHR,
//         DSN=SYS1.MACLIB
//SYSLIN   DD  DSN=&LOADSET(DSNTILMP),DISP=(NEW,PASS),
//             UNIT=SYSALLDA,
//             SPACE=(800,(50,50,2)),DCB=(BLKSIZE=800)
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT2   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT3   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSIN    DD  *
    DSN6ENV    MVS=XA
    DSN6SPRM   RESTART,                 X
               .
               .
               .
               AUTH=YES,
                                               X
               AUTHCACH=1024,                                         X
               BINDNV=BINDADD,                                        X
               BMPTOUT=4,                                             X
               CACHEDYN=YES,
                                           X
               .
               .
               .
               MAXKEEPD=16000,
                                         X
```

```
              .
              .
              .
   DSN6ARVP   ALCUNIT=CYL,                                          X
              .
              .
              .
   DSN6LOGP   DEALLCT=(0),                                          X
              .
              .
              .
   DSN6SYSP   AUDITST=NO,                                           X
              BACKODUR=5,                                           X
             CHKFREQ=16000000,
                                 X
              CONDBAT=400,                                          X
              CTHREAD=1200,                                         X
              DBPROTCL=PRIVATE,                                     X
              DLDFREQ=5,                                            X
              DSSTIME=5,                                            X
              EXTRAREQ=100,                                         X
              EXTRASRV=100,                                         X
              EXTSEC=NO,                                            X
              IDBACK=1800,
                                        X
              .
              .
              .
            //*
```

WebSphere Application Server tuning tips for use with DB2:
Prepared statement caching effects on DB2 for OS/390 JDBC cursor objects

WebSphere Application Server uses JDBC prepared statement caching as a performance enhancing feature. If you are using this feature together with DB2 for OS/390, be aware of the potential impact on the number of DB2 JDBC cursor objects available.

When you obtain a *ResultSet* object by executing a *PreparedStatement* object, a DB2 JDBC cursor object is bound to it until the corresponding DB2 prepared statement is closed. This happens when the DB2 *Connection* object is released from the WebSphere Application Server connection pool. From an application perspective, the result set, prepared statement, and connection are each closed in turn. However, the underlying DB2 *Connection* is pooled by the WebSphere Application Server, the underlying DB2 *PreparedStatement* is cached by the application server, and each underlying DB2 JDBC cursor object associated with each *ResultSet* created on this *PreparedStatement* object is not yet freed.

Each *PreparedStatement* object in the cache can have one or more result sets associated with it. If a result set is opened and not closed, even though you close the connection, that result set is still associated with the prepared statement in the cache. Each of the result sets has a unique JDBC cursor attached to it. This cursor is kept by the statement and is not released until the prepared statement is cleared from the WebSphere Application Server cache.

If there are more of the cached statements than there are cursors, eventually the execution of a *PreparedStatement* object results in the following exception:

```
java.sql.SQLException: DB2SQLJJDBCProfile Error: No more JDBC Cursors without hold
```

Some WebSphere Application Server tuning suggestions to help avoid this problem are:

1. Decrease the *statement cache size* setting on the DB2 for OS/390 data source definition. Setting this value to zero (0) eliminates statement caching, but causes a noticeable performance impact.
2. Decrease the *minConnections* connection pool setting on the DB2 for OS/390 data source definition.
3. Decrease the *Aged Timeout* connection pool setting on the DB2 for OS/390 data source definition. However, it is **NOT** recommended that you set this to zero (0), as this disables the Aged Timeout function.

RACF tuning tips for z/OS

Follow these guidelines for RACF tuning:

- Compared to no security, global security for a sample application took about 13%. Compared to no security, java2 security took about 18%. Compared to no security, global and java2 security took x%. Enabling a single role on every method in a sample application with global and Java2 security costs about 38%.
- As is always the case, don't turn things on unless you need them. In general, the cost of security has been highly optimized. However, if you don't need EJBROLEs, then don't enable the class in RACF.
- 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)
  ```
- 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.

TCP/IP tuning tips for z/OS

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

- 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:
  - Set MAXSOCKETS for the AF_INET filesystem high enough.
  - 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)
   .
```

- 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.
- 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.
- 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.

- 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
  ```

- 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
  ```

Tuning WebSphere MQ as a JMS provider

JMS tuning tips

If you use WebSphere MQ as a JMS provider to WebSphere Application Server, you can tune the setup by changing aspects of JMS and WebSphere MQ.

Non-persistent, non-durable and transaction-not-supported messages perform better, but at the cost that they are non-recoverable due to the lack of persistence.

- Turn off all tracing. Use the display trace command to get the trace number.

  ```
  D TRACE
  ```

  Then use the stop trace global command to turn off the specific trace number

  ```
  STOP TRACE (G) TNO(xxx)
  ```

- Create long-lived queue manager connections rather than creating and destroying connections on every message.

  Reuse connections and sessions.
- Use bindings mode if your queue manager and client both reside on the same z/OS image. Bindings mode is a cross-memory interface that eliminates the need for WebSphere MQ to call TCP/IP. It can be implemented by not setting the TransportType on the connection.
- Client acknowledgements required that an ack be received from the client before another message is sent. Auto acknowledgements are a better choice and reduce delays.
- Performance is better if the client and queue manager have the same CCSID (so the queue manager does not need to translate message headers) than if they have different CCSIDs.
- Small messages are best. Use of system resources and throughput is proportional to the size of WebSphere MQ messages. However, if you must send large

amounts of data, one larger message is preferable to multiple small ones. If using very large messages (for example, over 1MB) see "WebSphere MQ Tips".

- Persistent, transacted messages perform better than persisted non-transacted messages because multiple WebSphere MQ commits can be delayed until the end of the transaction.
- Express (nonpersistent) messages perform most optimally, so use them if your application does not require persistence.
- Application Server Framework (ASF) messaging generally adds more overhead then non-ASF messaging.
- WebSphere MQ local queues defined as DEFSOPT(SHARED) with the SHARE option, and shared by multiple threads or processes generally perform better than non-shared queues and use fewer resources.
- Message-driven beans are asynchronous by nature, therefore, should never be forced to run in a serial mode (Maxsession=1). Set a realistic number for Maxsession for the number of concurrent sessions.

WebSphere MQ tuning tips

- Turn off WebSphere MQ tracing in the MQ ZPARMS by specifying:.

```
TRACSTR=NO,         TRACING AUTO START               X
```

Example:

```
//*
//*             Assemble step for CSQ6LOGP
//*
//LOGP   EXEC PGM=ASMA90,PARM='DECK,NOOBJECT,LIST,XREF(SHORT)',
//            REGION=4M

//SYSLIB   DD DSN=MQSERIES.V5R3M0.SCSQMACS,DISP=SHR


//         DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&LOGP;,
//           UNIT=SYSDA,DISP=(,PASS),
//           SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
         CSQ6LOGP INBUFF=60,         LOG INPUT BUFFER SIZE (KB)        X
               MAXRTU=2,             MAX ALLOCATED ARCHIVE LOG UNITS   X
               DEALLCT=0,            ARCHIVE LOG DEALLOCATE INTERVAL   X
               MAXARCH=500,          MAX ARCHIVE LOG VOLUMES           X
               OFFLOAD=YES,          ARCHIVING ACTIVE                  X
               OUTBUFF=4000,         LOG OUTPUT BUFFER SIZE (KB)       X
               TWOACTV=YES,          DUAL ACTIVE LOGGING               X
               TWOARCH=YES,          DUAL ARCHIVE LOGGING              X
               TWOBSDS=YES,          DUAL BSDS                         X
               WRTHRSH=20            ACTIVE LOG BUFFERS
         END
/*
//*
//*             Assemble step for CSQ6ARVP
//*
//ARVP   EXEC PGM=ASMA90,COND=(0,NE),
//            PARM='DECK,NOOBJECT,LIST,XREF(SHORT)',
//            REGION=4M

//SYSLIB   DD DSN=MQSERIES.V5R3M0.SCSQMACS,DISP=SHR


//         DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&ARVP;,
//            UNIT=SYSDA,DISP=(,PASS),
//            SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
         CSQ6ARVP ALCUNIT=BLK,       UNITS FOR PRIQTY/SECQTY           X
               ARCPFX1=WIT.MQ1.WITA, DSN PREFIX FOR ARCHIVE LOG 1      X
               ARCPFX2=WIT.MQ2.WITA, DSN PREFIX FOR ARCHIVE LOG 2      X
               ARCRETN=9999,         ARCHIVE LOG RETENION TIME (DAYS)  X
               ARCWRTC=(1,3,4),      ARCHIVE WTO ROUTE CODE            X
               ARCWTOR=NO,           PROMPT BEFORE ARCHIVE LOG MOUNT   X
               BLKSIZE=24576,        ARCHIVE LOG BLOCKSIZE             X
               CATALOG=YES,          CATALOG ARCHIVE LOG DATA SETS     X
               COMPACT=NO,           ARCHIVE LOGS COMPACTED            X
               PRIQTY=4320,          PRIMARY SPACE ALLOCATION          X
               PROTECT=NO,           DISCRETE SECURITY PROFILES        X
               QUIESCE=5,            MAX QUIESCE TIME (SECS)           X
               SECQTY=540,           SECONDARY SPACE ALLOCATION        X
               TSTAMP=YES,           TIMESTAMP SUFFIX IN DSN           X
               UNIT=DASD,            ARCHIVE LOG DEVICE TYPE 1         X
               UNIT2=                ARCHIVE LOG DEVICE TYPE 2
         END
/*
//*
//*             Assemble step for CSQ6SYSP
//*
//SYSP   EXEC PGM=ASMA90,COND=(0,NE),
//             PARM='DECK,NOOBJECT,LIST,XREF(SHORT)',
//             REGION=4M

//SYSLIB   DD DSN=MQSERIES.V5R3M0.SCSQMACS,DISP=SHR


//         DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPUNCH DD DSN=&&SYSP;,
//            UNIT=SYSDA,DISP=(,PASS),
//            SPACE=(400,(100,100,1))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *

CSQ6SYSP CTHREAD=600,       TOTAL NUMBER OF CONNECTIONS
 X
               CMDUSER=CSQOPR,       DEFAULT USERID FOR COMMANDS       X
               EXITLIM=30,           EXIT TIMEOUT (SEC)                X
               EXITTCB=8,            NUMBER OF EXIT SERVER TCBS        X

IDBACK=500,            NUMBER OF NON-TSO CONNECTIONS
X
               IDFORE=100,           NUMBER OF TSO CONNECTIONS         X

LOGLOAD=900000,       LOG RECORD CHECKPOINT NUMBER
X
               OTMACON=(,,DFSYDRU0,2147483647,CSQ),  OTMA PARAMETERS   X
               QMCCSID=0,            QMGR CCSID                        X
               QSGDATA=(,,,),        QUEUE-SHARING GROUP DATA          X
               RESAUDIT=YES,         RESLEVEL AUDITING                 X
               ROUTCDE=1,            DEFAULT WTO ROUTE CODE            X
               SMFACCT=NO,           GATHER SMF ACCOUNTING             X
               SMFSTAT=NO,           GATHER SMF STATS                  X
               STATIME=30,           STATISTICS RECORD INTERVAL (MIN)  X
               TRACSTR=NO,           TRACING AUTO START                X

               TRACTBL=99,           GLOBAL TRACE TABLE SIZE X4K       X
               WLMTIME=30,           WLM QUEUE SCAN INTERVAL (SEC)     X
               SERVICE=0             IBM SERVICE USE ONLY
         END
/*
```

```
//*
//*  LINKEDIT CSQARVP, CSQLOGP and CSQSYSP into a
//*  system parameter module.
//*
//LKED   EXEC PGM=IEWL,COND=(0,NE),
//      PARM='SIZE=(900K,124K),RENT,NCAL,LIST,AMODE=31,RMODE=ANY'
//*
//*   OUPUT AUTHORIZED APF LIBRARY FOR THE NEW SYSTEM
//*   PARAMETER MODULE.
//*
//SYSLMOD  DD DSN=SYS1.WITA.LINKLIB,DISP=SHR
//SYSUT1   DD UNIT=SYSDA,DCB=BLKSIZE=1024,
//            SPACE=(1024,(200,20))
//SYSPRINT DD SYSOUT=*
//ARVP     DD DSN=&&ARVP;,DISP=(OLD,DELETE)
//LOGP     DD DSN=&&LOGP;,DISP=(OLD,DELETE)
//SYSP     DD DSN=&&SYSP;,DISP=(OLD,DELETE)
//*
//*   LOAD LIBRARY containing the default system
//*   parameter module (CSQZPARM).
//*
//OLDLOAD  DD DSN=MQSERIES.V520.SCSQAUTH,DISP=SHR
//SYSLIN   DD *
   INCLUDE SYSP
   INCLUDE ARVP
   INCLUDE LOGP
   INCLUDE OLDLOAD(CSQZPARM)
 ENTRY CSQZMSTR
 NAME CSQZPARM(R)                  Your system parameter module name
/*
```

- Similarly, you are advised to turn off tracing in the channel initiator. Unlike base WebSphere MQ tracing, this parameter cannot be enabled dynamically. To turn tracing back on for debug purposes, you need to reassemble your MQ XPARMS. Enabling channel initiator tracing can degrade your system by 5-10%.

  The variable to set is as follows:

  ```
  TRAXSTR=NO,       START TRACE AUTOMATICALLY  YES|NO
  ```

- For improved performance in the laboratory, the following WebSphere MQ tuning parameters required modification from their default settings.The difference in most cases was significant.
  - Optimize the number of concurrent queue manager connections

    ```
    CTHREAD parameter of CSQ6SYSP (Maximum number of  concurrent
                                   connections to MQ)
    IDBACK parameter of CSQ6SYSP (Maximum number of background concurrent
                                  threads connected to MQ)
    ```

    CTHREAD is the maximum number of simultaneous connections to the queue manager. It should be greater than, or equal to, the sum of IDFORE and IDBACK. IDFORE is the number of concurrent TSO connections to WebSphere MQ, and IDBACK is the number of concurrent background connections, which includes jms threads. If any or all of these parameters are too low, applications are unable to connect to the queue manager. These parameters are found in the CSQ6SYSP section of the MQ ZPARMS. You can update the MQ ZPARMS at any time; the updates take effect the next time you restart the queue manager.
  - Adjust the WebSphere MQ checkpoint interval and active log buffers

    ```
    LOGLOAD parameter of CSQ6SYSP (number of log records written
                                   before a checkpoint)
    WRTHRSH parameter of CSQ6LOGP (Number of active log buffers)
    ```

    When using persistent messages, it is particularly important to pay attention to logging characteristics. MQ logs should always be placed on high performance volumes with DASD fast write enabled. MQ logs are often the single most significant bottleneck when using persistent messages. The LOGLOAD parameter controls the number of log records written before a checkpoint (assuming, of course, that the log is large enough to hold this number of records). Checkpoints generally should occur no more frequently than every 5 or 6 minutes. If your MQ checkpoints are more frequent, you may need to increase either the size of the logs, the value of LOGLOAD, or both. In the laboratory, we use a LOGLOAD value of 900000 because we execute high throughput jms workloads that are very write-intensive. When the space on the log is exhausted, a log switch occurs which disrupts performance. You can avoid frequent log switches by increasing the size of the log(s).

WRTHRSH is the number of active log buffers, and determines how much data is held in memory before a log write occurs. If you have a high I/O rate to your log volume(s), you may wish to increase this parameter. In the laboratory, we use a value of 200.

Generally speaking, your message rate for persistent messages cannot exceed the bandwidth capacity of your slowest log volume. For example, if your were sending messages of 5KB at a throughput rate of 300 per second, you would be writing at least 1.9 MB of data per second to the log (this is roughly 1.3KB plus the user message size for each logged message).

– Specify the size of the archive logs

```
SECQTY parameter of CSQ6ARVP  (archive log space allocation)
PRIQTY parameter of CSQ6ARVP  (archive log space allocation)
ALCUNIT parameter of CSQ6ARVP  (archive Log allocation unit)
TWOARCH parameter of CSQ6LOGP  (dual archive logs)
```

The PRIQTY and SECQTY parameters control the size of the archive logs. Generally it is best to allocate them (ALCUNIT) in cylinders in lieu of blocks. Depending upon your data integrity requirements, you may or may not choose to have dual archive logging.

– Specify the number of buffers

```
BUFFERS  (number of buffers on the DEFINE BUFFERPOOL statement)
```

Use the DEFINE BUFFERPOOL statement to specify the number of buffers. It is important to insure the number of buffers is large enough to hold at least an entire message (and its headers). Otherwise, WebSphere MQ is forced to write to the Page Data sets for every message. For example, a 100MB message requires at least 26000 pages in the bufferpool. MQ Buffer Manager statistics can be used to determine the number of times a buffer was unavailable. See Support Pac MP1B, MQSeries for OS/390 V5.2 - Interpreting accounting and statistics data.

– Specify queue definitions

```
INDXTYPE(NONE)  (index specification for queue definitions) or
INDXTYPE(CORRELID)
DEFPSIST(NO)
```

Unless your applications retrieve messages by other than correlation ID (which is the case for jms publish/subscribe) or message ID, it is normally best not to specify message selectors on queue definitions. You should, however, make sure you have specified INDXTYPE(CORRELID) on the SYSTEM.JMS.ND/D queues, and/or on the SYSTEM.JMS.ND.CC / .D.CC queues, and/or on any shared message queues for publish/subscribe. Specify DEFPSIST(NO) unless you want messages on a particular queue to default to persistent. Because DEFPSIST(YES) affects performance, make sure you really want persistent messages.

If your applications use message ID to access the message, we highly recommend indexing the queue on message ID. You can add the index either at the time of queue definition or on the fly by using the WebSphere MQ command INDXTYPE(MSGID).

– Specify channel definitions

```
BATCHSZ parameter for queue definitions (Number of messages
                                         sent as a batch)
```

We did not modify the BATCHSZ parameter in the laboratory.

GRS tuning tips for z/OS

WebSphere for z/OS uses GRS to communicate information between servers in a sysplex. When there are multiple servers defined in a system or a sysplex, a request may end up on the wrong server. To determine where the transaction is running WebSphere uses GRS. Therefore, if you are using global transactions, WebSphere will issue an enqueue for that transaction at the start of the transaction and hold on to that enqueue until the transaction ends. WebSphere for z/OS uses GRS enqueues for the following:

- Two-phase commit transactions involving more than one server
- HTTP sessions in memory
- Stateful EJBs
- ″Sticky″ transactions to keep track of pseudo-conversational states.
- If you **are not** in a sysplex, you should configure GRS=NONE.
- If you **are** in a sysplex, we strongly recommend GRS=STAR.

This requires configuring GRS to use the coupling facility. All of the WebSphere enqueues are issued with RNL=NO, which prevents misconfiguring the GRSRNLxx with inappropriate values. See the GRS documentation for details on setting this up.

Java virtual machine (JVM) tuning tips for z/OS

Before you begin:

1. Ensure that you have the most recent version of JVM that is supported by WebSphere for z/OS.
2. Have the most recent PTFs, since almost every PTF level has improved performance of the JVM.
3. Have sufficient JVM Heap Size. Refer to “Java virtual machine storage tuning tips for z/OS” on page 22 for a discussion of this setting.

How to view or set: Use the WebSphere Administrative console :

1. Click **Servers > Application Servers >** *server_name* **> Process Definition > Java Virtual Machine**
2. Select the options that are listed on the Configuration tab.

- Run with the JIT (Just In Time) compiler active.

  In the General Properties section of the Configuration Tab, ensure that Disable JIT is not selected. The default is JIT support enabled.
- Do not specify the debug version of the JVM `libjava_g` in your libpath. Severe performance degradation is likely when running with the debug version of the JVM.
- Have Classpath point to only the classes you need (the classes that are referenced most frequently should be located near the front of the path, if possible).

  In the General Properties section of the Configuration Tab, enter the Classpath in the text box of the Classpath option.

- Verify the Classpath as part of the Java configuration.
- 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
  ```

  For JDK Version 1.4.1 the specification reduces servant startup time at the cost of some runtime performance. For JDK Version 1.3.1, these options can reduce servant startup time by as much as 40%. However, they will reduce runtime throughput by about 8%.
- 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.

  Note: Please see the ″Applications″ section in the WebSphere Application Server for z/OS information center, access to which can be obtained through the WebSphere for z/OS library Web site http://www.ibm.com/software/webservers/appserv/zos_os390/library.html for more information on ″Application client troubleshooting tips.″

CICS tuning tips for z/OS

These recommendations only apply to WebSphere applications that access CICS.

The LGDFINT system initialization parameter specifies the log defer interval used by CICS log manager when determining how long to delay a forced journal write request before invoking the MVS system logger. The value is specified in milliseconds. Performance evaluations of typical CICS transaction workloads have shown that the default setting of 5 milliseconds gives the best balance between response time and central processor cost. Be aware that CICS performance can be adversely affected by a change to the log defer interval value. Too high a value will delay CICS transaction throughput due to the additional wait before invoking the MVS system logger. An example of a scenario where a reduction in the log defer interval might be beneficial to CICS transaction throughout would be where many forced log writes are being issued, and little concurrent task activity is occurring. Such tasks will spend considerable amounts of their elapsed time waiting for the log defer period to expire. In such a situation, there is limited advantage in delaying a call to the MVS system logger to write out a log buffer, since few other log records will be added to the buffer during the delay period.

- Set the LGDFINT system initialization parameter to 5.

  While CICS is running, you can use the CEMT SET SYSTEM[LOGDEFER(*value*)] command to alter the LGDFINT setting dynamically.
- Set the CICS RECEIVECOUNT value high enough to handle all concurrent EXCI pipes on the system.

  The default value is 4. You set this value in the EXCI sessions resource definition.

For more detailed information on CICS, refer to the *CICS Performance Guide*.

Tuning the WebSphere Application Server for z/OS runtime

Steps involved in tuning the WebSphere Application Server for z/OS runtime to optimize performance include reviewing the:

- "Review the WebSphere for z/OS configuration"
- "Internal tracing tips for WebSphere for z/OS"
- "Location of executable programs tips for z/OS" on page 45
- "Security tuning tips for z/OS" on page 45
- Web server plug-in tuning tips in the *Setting up the application serving environment* PDF book.

Review the WebSphere for z/OS configuration

The first thing to do is 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. When each server starts, the runtime prints out the current configuration data in the joblog.

Internal tracing tips for WebSphere for z/OS

WebSphere 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.
- If you use any level of tracing, including ras_trace_defaultTracingLevel=1, ensure that you set ras_trace_outputLocation to BUFFER.

  ras_trace_defaultTracingLevel=1 will write exceptions 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_defaultTracingLevel=3, you may experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you may only experience a 15% degradation in throughput.
- Set the ras_trace_BufferCount=4 and ras_trace_BufferSize=128.

  This will get 512KB of storage for the trace buffers (the minimum allowed) and reduce memory requirements.
- 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

The next thing to review in the configuration is where your program code is located. IBM recommends that you install as much of the WebSphere for z/OS code in LPA as is reasonable, and the remainder in the linklist. This ensures that you have eliminated any unnecessary steplibs which can adversely 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" on page 28 for USS shared file system tuning considerations.

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

Security tuning tips for z/OS

As a general rule, two things happen when you increase security: the cost per transaction increases and throughput decreases.

By default, WebSphere Application Server runs with security off. With security turned off there is virtually no overhead. However, we recommend that you run with security enabled, even though the runtime will always incur a small price to collect and carry the security credential information for users and the server.

- 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. Since all 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.
- 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.
- If you do not need Java 2 security, you should disable it.

  For instructions on how to disable Java 2 security, refer to the *Securing applications and their environment* PDF book.
- Use the lowest level of authorization consistent with your security needs.

  You have several 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.
- If using SSL, select the lowest level of encryption consistent with your security requirements.

  WebSphere allows 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.

Tuning for J2EE applications

Steps involved in tuning the J2EE applications performance include:
- "Topology planning and performance"
- "J2EE container and applications" on page 47
- "J2EE application programming tips" on page 48
- "Tuning for SOAP" on page 48
- "Throttling of inbound message flow for non-JCA 1.5 message-driven beans" on page 50

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 for z/OS.

- Single server or multiple servers?

  WebSphere 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 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 the *Setting up the application serving environment* PDF book.

  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.

J2EE container and applications

In WebSphere Application Server, there are several types of enterprise beans and several transaction policies supported. Selection of each type has performance implications. While we won't be able to give an exhaustive treatise on this yet, we will give some basic rules.

Enterprise bean development performance ramifications: There are three basic bean types in WebSphere Application Server: session, entity, and message driven.

- Session beans

  Within a session bean in WebSphere Application Server, there are stateless and stateful session beans.

  Stateless session bean

  The lowest overhead type of bean. They are cheap to create, do very little automatically and, if not cleaned up by the application, will go away when the server terminates.

  Stateful session bean

  The default for a stateful session bean is not to fortify its state to a backend server except in the case of a controlled server shut down. In this configuration, a stateful bean is slightly more overhead than a stateless bean. The configuration overhead of hardening the stateful bean state at the end of each transaction has yet to be quantified.
- Entity beans

  In WebSphere Application Server, entity beans come in two flavors: bean managed persistence (BMP) and container managed persistence (CMP).

  Since managing persistence is the responsibility of the bean in BMP, it really depends on the way the load and store is implemented whether a BMP is faster than a CMP. CMP beans manage persistence. The CMP bean implementation is highly optimized and will often produce better performance than a typical BMP bean. Additional improvements are expected which will add even more flexibility to CMP beans.
- Message driven beans (MDBs). See Using message-driven beans to automatically retrieve messages.

- Marking a bean method with the `readonly` extended deployment descriptor will cause the runtime to avoid writing the state of the bean back out. Specify this value on a method when you know that the method does not update any attributes of the bean.

Transaction policy tuning tips: There are seven transaction policies in WebSphere Application Server:
- TRANSACTION_REQUIRES
- TRANSACTION_REQUIRES_NEW
- TRANSACTION_SUPPORTS
- TRANSACTION_NOT_SUPPORTED
- TRANSACTION_BEAN_MANAGED
- TRANSACTION_NEVER
- TRANSACTION_MANDATORY

These transaction policies control how enterprise beans are associated with global or local transactions. Generally, local transactions are the fastest.

J2EE application programming tips

These programming tips relate to the following topics:
- **JavaServer Pages (JSPs)**
  - Disable session state of JSPs.

    <%@ page language=″java″ contentType=″text/html″ session=″false″ %>
  - By default, JSPs will save session state

    <%@ page language=″java″ contentType=″text/html″ %>
  - Replace setProperties() calls in your JSPs with direct calls to the appropriate setxxx methods.
- **Java Database Connectivity (JDBC)**
  - Make sure you are at the current JDBC level.
  - Use prepared statements to allow dynamic statement cache of DB2 on z/OS.
  - Don't include literals in the prepared statements, use a parameter marker ″?″ to allow dynamic statement cache of DB2 on z/OS.
  - Use the right getxxx method by each data type of DB2.
  - Turn auto commit off when just read-only operations are performed.
  - Use explicit connection context objects.
  - When coding an iterator, you have a choice of named or positioned. For performance, we recommend positioned iterators.
  - Close prepared statements before reusing the statement handle to prepare a different SQL statement within the same connection.
  - As a bean developer, you have the choice of JDBC or SQLJ. JDBC makes use of dynamic SQL whereas SQLJ generally is static and uses pre-prepared plans. SQLJ requires an extra step to create and bind the plan whereas JDBC does not. SQLJ, as a general rule, is faster than JDBC.
  - With JDBC and SQLJ, you are better off writing specific calls that retrieve just what you want rather than generic calls that retrieve the entire row. There is a high per-field cost.

Tuning for SOAP

The Simple Object Access Protocol (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 allow 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 the Object Request Broker service settings topic in the *Setting up the application serving environment* PDF book.

- 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) may impact system resources like memory.
- You may need to increase the default Java heap size. SOAP and XML (DOM) are storage-intensive and small heap sizes may 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.
- Consider writing your own serializers and deserializers, avoiding reflection.
- Consider writing a servlet to use a SAX parser rather than the SOAP runtime if you require improved performance. Alternately, you can download and install Apache AXIS, or use the Web Services Technology Preview.

SOAP V2.3 in WebSphere Application Server: SOAP 2.3 has a number of enhancements over SOAP 2.2. They are described in full in http://ws.apache.org/soap/releases.html#v2.2. Some of the key enhancements in WebSphere V 5.0 are as follows:
- Support for document-oriented messaging
- Web Services Definition 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, for integration of decentralized distributed environment, where the clients have little knowledge of the application implementation. 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.

Throttling of inbound message flow for non-JCA 1.5 message-driven beans

This topic describes how to throttle message delivery for endpoint message-driven beans deployed on a JMS provider that does not have a JCA 1.5 resource adapter (such as the V5 Default Messaging and WebSphere MQ).

Message driven bean (MDB) throttle support controls the flow of work requests (WRs) into the WLM queue. The WRs result from messages being placed into a JMS destination. The MDB throttle allows or blocks a WR from being queued based on the maximum sessions set for the Listener Port and the current amount of in-flight requests. For more information on MDB throttle support, refer to “MDB throttle support” on page 52.

Configuring MDB throttle support involves listener port settings, connection factory settings, and MQ property settings. The proper setting of these properties will result in a configuration that handles message processing satisfactorily.

1. Set the listener port maximum sessions property. 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*****-> 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. A good initial value for this setting is 100. The throttle also uses this value to compute the low threshold value. The

actual number of server sessions is limited by the number of worker threads in the servant region and by the connection factory session pool max connections property.

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. This property may be set to the total amount of connections that will be created. We recommend setting it to at least the number of worker threads available to the servant region.
   e. Under Additional Properties of the Connection Factory, activate the Session Pool panel.
   f. Set the Max Connections property for the Session Pool. This property may be set to the total amount of sessions that will be created or used concurrently from a single connection. This allows sessions to be readily available and reduces the probability of having to wait for a session to become available in order to do message processing. If the value is less than the Listener Port Maximum sessions property and the number of worker threads, then the throttle might request more sessions than there are available. This would result in one or more requests having to wait for a session to become free, which could have an undesirable effect on concurrent processing of requests.
3. Set the MQ related properties. The properties discussed in this section pertain to properties that have an effect on the number of messages in the destination and the number of connections to the queue manager. These properties include the MQ Backout Count Threshold (BOTHRESH), IDBACK, and IDFORE. For more information on these MQ settings, refer to:

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

   WebSphere MQ Problem Determination Guide (GC34-6054-01).
   a. Set the MQ Backout Count Threshold (BOTHRESH) to a value less than the Listener Port Maximum retries property. If the BOTHRESH value is greater than the Listener Port Maximum retries value, then the Listener Port will be stopped for excessive message roll backs as in the case of poison messages, and the corrupted message will still remain in the destination.

      Setting this property to a value less than the Listener Port Maximum retries value prevents the same message from remaining in the destination after the message has been rolled back the specified number of times. Reaching this count threshold causes the message to be placed into the backout queue, which helps reduce the number of messages in the destination. Only the required number of Work Requests are created for the healthy messages, therefore improving resources utilization and performance.
   b. Set CTHREAD to the maximum number of connections from batch, CICS, IMS, and TSO tasks to a single instance of WebSphere MQ. This can be thought of as the total of connections available to WebSphere MQ. Make sure that the total of the IDBACK and IDFORE properties discussed in the next two substeps do not exceed this value.
   c. Set IDBACK to the maximum number of connections to a single instance of WebSphere MQ from batch or TSO background tasks. The number of

connections to the queue manager impact performance. Choose the minimal number of connections that allows satisfactory message processing.

d. Set IDFORE to the maximum number of connections to a single instance of WebSphere MQ from TSO foreground tasks. The number of connections to the queue manager impact performance. Choose the minimal number of connections that allows satisfactory message processing.

To set the MDB throttle to allow 100 messages to be queued before blocking message retrieval on a server with 40 Worker Threads, two connections (one for serving the MDBs and one per MDB instance for sending a reply message from the MDB's onMessage method), and a Listener Port maximum retries set to 5, set the following properties:

1. Set the Listener Port Maximum sessions property to 100. This automatically sets the throttle High Threshold to 100.
2. Set the Connection Factory's Connection Pool to have enough connections by setting the Max Connections property to the minimum of both the Maximum sessions and the Max Connections, 40 in this case.

   The number of worker threads limits the amount of server sessions available. Therefore, assuming the advanced ORB setting of LONGWAIT (40 threads), the server session pool will be set to allow at most 40 server sessions to be created.
3. Set the Max Connections property for both the Connection Factory's Connection Pool and Session Pool to have enough sessions by setting the Max Connections property to the amount of concurrent sessions to be used during message processing.

   In this example, there can be at most 40 MDB instances, and a total of 1 session is needed per MDB instance, then the property should be set to 40.
4. Set the MQ BOTHRESH property to less than the Listener Port Maximum retries property.
5. Set the CTHREAD value higher than 40.

   For the CTHREAD, IDBACK, and IDFORE the total number of connections and how they are used must be taken into consideration. In this example, at most 40 connections could be made to WebSphere MQ from the Connection Factory, so CTHREAD should be higher than this value.

For debugging tips, refer to "Best practices for debugging MDB throttle support" on page 53.

MDB throttle support:

The reasoning behind any throttling mechanism is to control the flow of the resource at hand from one stage to another. Satisfactory message retrieval should be obtained by adjusting the rate at which message reference (MRs) are handled and work requests (WRs) are created. Without any type of control over the amount of WRs that are queued into the WLM queue, an excessive amount of WRs accumulate and are less likely to get processed by a minimal number of resources in the Servant Region. Control is based on available resources.

The messaging support provided by WebSphere Application Server allows message processing to be split across regions. The messaging component in the WAS Controller Region is in charge of processing the Message References generated by MQ after messages are placed in the destination. The messaging component in the WAS Servant Region has the responsibility of retrieving the messages from the destination and deliver them to the MDB. WLM is used to queue work requests related to the Message References received in the Controller Region. This architecture poses problems when a large number of messages are placed in the destination queue.

Messaging flow. A message is sent to a destination by a JMS or MQ client. The destination is browsed for messages. Once the message is detected, a Message Reference (MR) is created and sent to the Message Reference Handler (MRH). The MRH calls the required service to queue the Work Request (WR) into the WLM queue. The work is dispatched in the Servant Region. The MR is consumed by the Listener causing the actual message in the destination to be retrieved. The message is presented to the MDB. The message is acknowledged and removed from the destination after the MDB is done processing it.

A considerably large number of messages sent to the destination causes at least the same amount of WRs to be queued into the WLM queue. If the resources allocated in the Servant Region are not sufficient to process the large amount of work presented, messages might get browsed two or more times after a certain amount of time. This worsens the situation by scheduling more WRs for the same messages, causing frequent message re-delivery.

Messaging flow with the throttle support. A message is sent to a destination. An MR is created and sent to the MRH. The MRH notifies the throttle of the message reference. The Throttle allows or blocks a WR from being queued based on the Maximum sessions set for the Listener Port and the current amount of in-flight requests.

When a high threshold is met, the throttle blocks queue browsing. Message processing continues and current requests are dispatched to available Servant regions. When the low threshold is met, the throttle allows more WRs to be queued.

Best practices for debugging MDB throttle support: 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.

Tuning Java virtual machines

The application server, being a Java process, requires a Java virtual machine (JVM) to run, and to support the Java applications running on it. As part of configuring an application server, you can fine-tune settings that enhance system use of the JVM. In addition to the following tuning parameters, see also Java memory tuning tips.

A JVM provides the runtime execution environment for Java based applications. WebSphere Application Server is a combination of a JVM runtime environment and a Java based server runtime. It can run on JVMs from different JVM providers. To determine the JVM provider on which your Application Server is running, issue the java –fullversion command from within your WebSphere Application Server `install_root/java/bin` directory. You can also check the SystemOut.log from one of your servers. When an application server starts, Websphere Application Server writes information about the JVM, including the JVM provider information, into this log file.

From a JVM tuning perspective, there are two main types of JVMs:
- IBM JVMs
- Sun HotSpot based JVMs, including Sun HotSpot JVM on Solaris and HP's JVM for HP-UX

Even though JVM tuning is dependent on the JVM provider general tuning concepts 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 run-time.
- Java memory or heap tuning. The JVM memory management function, or garbage collection provides one of the biggest opportunities for improving JVM performance.
- Class loading tuning.
- **Optimize the startup performance and the runtime performance** In some environments, it is more important to optimize the startup performance of your WebSphere Application Server rather than the runtime performance. In other environments, it is more important to optimize the runtime performance. By default, IBM JVMs are optimized for runtime performance while HotSpot based JVMs are optimized for startup performance.

The behavior of the Java JIT compiler has the biggest influence over rather startup or runtime performance are optimized. The initial optimization level used by the compiler influences the length of time it takes to compile a class method and the length of time it takes to start the server. For faster startups, you can reduce the initial optimization level that the compiler uses. This means that the runtime performance of your applications may be degraded because the class methods are now compiled at a lower optimization level.

It is hard to provide a specific runtime performance impact statement because the compilers might recompile class methods during runtime execution based upon the compiler's determination that recompiling might provide better performance. Ultimately, the size of the application will have a major influence on the amount of runtime degradation. Smaller applications have a higher probability of having their methods recompiled. Bigger applications are less likely to have their methods recompiled. The default settings for IBM JVMs use a high optimization level for the initial compiles. You can use the following IBM JVM option if you need to change this behavior:

-Xquickstart

This setting influences how the IBM JVM uses a lower optimization level for class method compiles, which provides for faster server startups, at the expense of runtime performance. If this parameter is not specified, the IBM JVM defaults to starting with a high initial optimization level for compiles. This setting provides faster runtime performance at the expense of slower server starts.

Default:	High initial compiler optimizations level
Recommended:	High initial compiler optimizations level
Usage:	-Xquickstart can provide faster server startup times.

JVMs based on Sun's Hotspot technology initially compile class methods with a low optimization level. Use the following JVM option to change this behavior:

-server

JVMs based on Sun's Hotspot technology initially compile class methods with a low optimization level. These JVMs use a simple complier and an optimizing JIT compiler. Normally the simple JIT compiler is used. However you can use this option to make the optimizing compiler the one that is used. This change will significantly increases the performance of the server but the server takes longer to warm up when the optimizing compiler is used.

Default:	Simple compiler
Recommended:	Optimizing compiler
Usage:	-server enables the optimizing compiler.

- **Set the heap size** The following command line parameters are useful for setting the heap size.
  - **-Xms**

    This setting controls the initial size of the Java heap. Properly tuning this parameter reduces the overhead of garbage collection, improving server response time and throughput. The default setting for this option is typically too low, resulting in a high number of minor garbage collections

Default:	50MB
Recommended:	Workload specific, but higher than the default.

Usage:	-Xms256m sets the initial heap size to 256 megabytes

– **-Xmx**

This setting controls the maximum size of the Java heap. Properly tuning this parameter can reduce the overhead of garbage collection, improving server response time and throughput. The default setting for this is typically too low, resulting in a high number of minor garbage collections.

Default:	256MB
Recommended:	Workload specific, but higher than the default.
Usage:	-Xmx512m sets the maximum heap size to 512 megabytes

– **-Xlp**

This setting can be used with the IBM JVM to allocate the heap using large pages. However, if you use this setting your operating system must be configured to support large pages. Using large pages can reduce the CPU overhead needed to keep track of heap memory and might also allow the creation of a larger heap.

- **Tune the IBM JVM's garbage collector**

  Use the Java -X option to see the list of memory options.

  – **-Xgcpolicy**

  Setting gcpolicy to optthruput disables concurrent mark. If you do not have pause time problems, denoted by erratic application response times, you should get the best throughput using this option. Setting gcpolicy to optavgpause enables concurrent mark with its default values. This setting alleviates erratic application response times caused by normal garbage collection. However, this option might decrease overall throughput.

Default:	optthruput
Recommended:	optthruput
Usage:	Xgcpolicy:optthruput

– **-Xnoclassgc**

By default the JVM unloads a class from memory when there are no live instances of that class left, but this can degrade performance. Turning off class garbage collection eliminates the overhead of loading and unloading the same class multiple times.

If a class is no longer needed, the space that it occupies on the heap is normally used for the creation of new objects. However, if you have an application that handles requests by creating a new instance of a class and if requests for that application come in at random times, it is possible that when the previous requester is finished, the normal class garbage collection will clean up this class by freeing the heap space it occupied, only to have to re-instantiate the class when the next request comes along. In this situation you might want to use this option to disable the garbage collection of classes.

Default:	class garbage collection enabled
Recommended:	class garbage collection disabled
Usage:	Xnoclassgc disables class garbage collection

For additional information, see the following DeveloperWorks article:

http://www.ibm.com/developerworks/java/

- **Tune the HP JVM's garbage collector**

  The HP JVM relies on generational garbage collection to achieve optimum performance. The following command line parameters are useful for tuning garbage collection.

  – **-Xoptgc**

    This setting optimizes the JVM for applications with many short-lived objects. If this parameter is not specified, the JVM usually does a major (full) garbage collection. Full garbage collections can take several seconds and can significantly degrade server performance.

Default:	off
Recommended:	on
Usage:	-Xoptgc enables optimized garbage collection.

  – **-XX:SurvivorRatio**

    The Java heap is divided into a section for old (long lived) objects and a section for young objects. The section for young objects is further subdivided into the section where new objects are allocated (eden) and the section where new objects that are still in use survive their first few garbage collections before being promoted to old objects (survivor space). Survivor Ratio is the ratio of eden to survivor space in the young object section of the heap. Increasing this setting optimizes the JVM for applications with high object creation and low object preservation. Since WebSphere Application Server generates more medium and long lived objects than other applications, this setting should be lowered from the default.

Default:	32
Recommended:	16
Usage:	-XX:SurvivorRatio=16

  – **-XX:PermSize**

    The section of the heap reserved for the permanent generation holds all of the reflective data for the JVM. This size should be increased to optimize the performance of applications which dynamically load and unload a lot of classes. Specifying a value of 128 megabytes eliminates the overhead of increasing this part of the heap.

Default:	0
Recommended:	128 megabytes
Usage:	-XX:PermSize=128m sets PermSize to 128 megabytes

  – **-XX:+ForceMmapReserved**

    By default the Java heap is allocated ″lazy swap.″ This saves swap space by allocating pages of memory as needed, but this also forces the use of 4KB pages. This allocation of memory can spread the heap across hundreds of thousands of pages in large heap systems. This command disables ″lazy swap″ and allows the operating system to use larger memory pages, thereby optimizing access to the memory making up the Java heap.

Default:	off
Recommended:	on
Usage:	-XX:+ForceMmapReserved will disable ″lazy swap″.

– **-Xmn**

This setting controls how much space the young generation is allowed to consume on the heap. Properly tuning this parameter can reduce the overhead of garbage collection, improving server response time and throughput. The default setting for this is typically too low, resulting in a high number of minor garbage collections.

Default:	No default
Recommended:	Approximately 3/4 of the total heap size
Usage:	-Xmn768m sets the size to 768 megabytes

– **Virtual Page Size**

Setting the Java virtual machine instruction and data page sizes to 64MB can improve performance.

Default:	4MB
Recommended:	64MB
Usage:	Use the following command. The command output provides the current operating system characteristics of the process executable: `chatr +pi64M +pd64M /opt/WebSphere/AppServer/java/bin/PA_RISC2.0/native_threads/java`

– **-Xnoclassgc**

By default the JVM unloads a class from memory when there are no live instances of that class left, but this can degrade performance. Turning off class garbage collection eliminates the overhead of loading and unloading the same class multiple times.

If a class is no longer needed, the space that it occupies on the heap is normally used for the creation of new objects. However, if you have an application that handles requests by creating a new instance of a class and if requests for that application come in at random times, it is possible that when the previous requester is finished, the normal class garbage collection will clean up this class by freeing the heap space it occupied, only to have to re-instantiate the class when the next request comes along. In this situation you might want to use this option to disable the garbage collection of classes.

Default:	class garbage collection enabled
Recommended:	class garbage collection disabled
Usage:	Xnoclassgc disables class garbage collection

For additional information on tuning the HP virtual machine, see Java technology software HP-UX 11i.

- **Tune the Sun JVM's garbage collector**

On the Solaris platform, the WebSphere Application Server runs on the Sun Hotspot JVM rather than the IBM JVM. It is important to use the correct tuning parameters with the Sun JVM in order to utilize its performance optimizing features.

The Sun HotSpot JVM relies on generational garbage collection to achieve optimum performance. The following command line parameters are useful for tuning garbage collection.

– **-XX:SurvivorRatio**

  The Java heap is divided into a section for old (long lived) objects and a section for young objects. The section for young objects is further subdivided into the section where new objects are allocated (eden) and the section where new objects that are still in use survive their first few garbage collections before being promoted to old objects (survivor space). Survivor Ratio is the ratio of eden to survivor space in the young object section of the heap. Increasing this setting optimizes the JVM for applications with high object creation and low object preservation. Since WebSphere Application Server generates more medium and long lived objects than other applications, this setting should be lowered from the default.

Default:	32
Recommended:	16
Usage:	-XX:SurvivorRatio=16

– **-XX:PermSize**

  The section of the heap reserved for the permanent generation holds all of the reflective data for the JVM. This size should be increased to optimize the performance of applications that dynamically load and unload a lot of classes. Setting this to a value of 128MB eliminates the overhead of increasing this part of the heap.

Recommended:	128 MB
Usage:	XX:PermSize=128m sets perm size to 128 megabytes.

– **-Xmn**

  This setting controls how much space the young generation is allowed to consume on the heap. Properly tuning this parameter can reduce the overhead of garbage collection, improving server response time and throughput. The default setting for this is typically too low, resulting in a high number of minor garbage collections. Setting this setting too high can cause the JVM to only perform major (or full) garbage collections. These usually take several seconds and are extremely detrimental to the overall performance of your server. You must keep this setting below half of the overall heap size to avoid this situation.

Default:	2228224 bytes
Recommended:	Approximately 1/4 of the total heap size
Usage:	-Xmn256m sets the size to 256 megabytes.

– **-Xnoclassgc**

By default the JVM unloads a class from memory when there are no live instances of that class left, but this can degrade performance. Turning off class garbage collection eliminates the overhead of loading and unloading the same class multiple times.

If a class is no longer needed, the space that it occupies on the heap is normally used for the creation of new objects. However, if you have an application that handles requests by creating a new instance of a class and if requests for that application come in at random times, it is possible that when the previous requester is finished, the normal class garbage collection will clean up this class by freeing the heap space it occupied, only to have to re-instantiate the class when the next request comes along. In this situation you might want to use this option to disable the garbage collection of classes.

Default:	class garbage collection enabled
Recommended:	class garbage collection disabled
Usage:	Xnoclassgc disables class garbage collection

For additional information on tuning the Sun JVM, see Performance Documentation for the Java HotSpot VM.

- **Tune HP's JVM for HP-UX** Set the following options to improve application performance:

```
-XX:SchedulerPriorityRange=SCHED_NOAGE
-Djava.nio.channels.spi.SelectorProvider=sun.nio.ch.DevPollSelectorProvider
-XX:-ExtraPollBeforeRead
```

Tuning transport channel services

The transport channel services manage client connections and I/O processing for HTTP and JMS requests. These new I/O services are based on new non-blocking I/O features available in Java™. These services provide a highly scalable foundation to WebSphere Application Server request processing.

Key features of the new transport channel services include:

- Scalability, which enables the WebSphere Application Server 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 TCP, HTTP, or Web container transport channels associated with a transport chain can improve the performance of that chain.

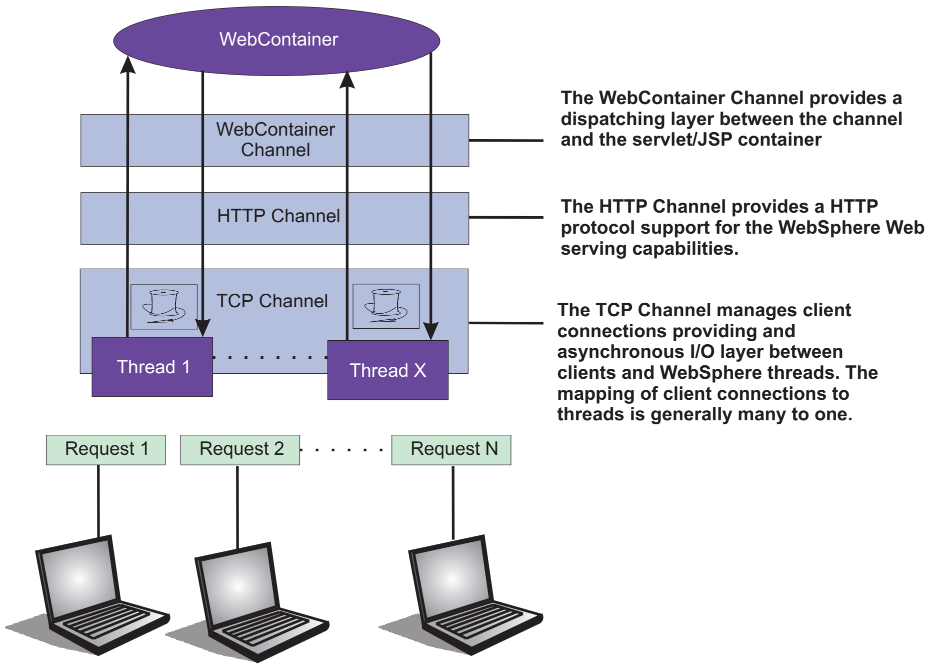
WebContainer
WebContainer Channel
HTTP Channel
TCP Channel
Thread 1
Thread X
Request 1
Request 2
Request N
The WebContainer Channel provides a dispatching layer between the channel and the servlet/JSP container
The HTTP Channel provides a HTTP protocol support for the WebSphere Web serving capabilities.
The TCP Channel manages client connections providing and asynchronous I/O layer between clients and WebSphere threads. The mapping of client connections to threads is generally many to one.

Figure 1. Transport Channel Service

- **Adjust TCP transport channel settings.** In the administration console, click **Servers > 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. **Leave the Maximum open connections parameter set to the default value.** This parameter controls the maximum number of connections that are available for a server's use. It should be left at the default value of 20000, which is the maximum number of connections allowed. The transport channel service by default manages high client connection counts and requires no tuning.
  4. **If client connections are being closed without data being written back to the client, change the value specified for the Inactivity timeout parameter.** This parameter controls the maximum number of connections available for a server's use. Upon receiving a new connection, the TCP transport channel waits for enough data to arrive to dispatch the connection to the protocol specific channels above the TCP transport channel. If not enough data is received during the time period specified for the Inactivity timeout parameter, the TCP transport channel closes the connection.

     The default value for this parameter is 60 seconds, which is adequate for most applications. You should increase the value specified for this parameter if your workload involves a lot of connections and all of these connections can not be serviced in 60 seconds.
  5. **Tune the size of your thread pools.** By default, a thread pool can have a minimum of 10 threads and a maximum of 50 maximum threads. To adjust these values, click on **Thread pools >***threadpool_name* and adjust the values specified for the Minimum Size and Maximum Size parameters for that thread pool.

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

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

- **Adjust HTTP transport channel settings.** In the administration console, click **Servers > 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 and is the optimal setting in most cases.

     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 (keep-alive) 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 Web container transport channel settings.** In the administration console, click **Servers > 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 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 will be 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.
- Click **Apply** and then **Save** to save these changes.

Checking hardware configuration and settings

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

These 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 (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 256MB 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 256MB of memory for each processor and 512 MB per application server. Some applications might require more memory.
- **Run network cards and network switches at full duplex**
  - **Description:** Run network cards and network switches at full duplex and use the highest supported speed. Full duplex is much faster than half duplex. Verify that the network speed of adapters, cables, switches, and other devices can accommodate the required throughput. Some Web sites might require multiple gigabit links.
  - **Recommendation** Make sure that the highest speed is in use on 10/100/1000 Ethernet networks.

Chapter 7. 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 and Service Oriented Architecture (SOA) • Web services security	**J2EE resources** • Data access resources • Messaging resources • Mail, URLs, and more **WebSphere extensions** • ActivitySessions • Application profiling • Asynchronous beans • Dynamic caching • Dynamic and EJB query • Internationalization • Object pools • Scheduler • Startup beans • Work area

Web applications

Tuning session management

WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

The table summarizes the features, including whether they apply to sessions tracked in memory, in a database, with memory-to-memory replication, or all. Click a feature for details about the feature. Some features are easily manipulated using administrative settings; others require code or database changes.

Feature or option	**Goal**	**Applies to sessions in memory, database, or memory-to-memory**
Write frequency	Minimize database write operations.	Database and Memory-to-Memory
Session affinity	Access the session in the same application server instance.	All
Multirow schema	Fully utilize database capacities.	Database
Base in-memory session pool size	Fully utilize system capacity without overburdening system.	All
Write contents	Allow flexibility in determining what session data to write	Database and Memory-to-Memory
Scheduled invalidation	Minimize contention between session requests and invalidation of sessions by the Session Management facility. Minimize write operations to database for updates to last access time only.	Database and Memory-to-Memory
Tablespace and row size	Increase efficiency of write operations to database.	Database (DB2 only)

Configuring scheduled invalidation

Instead of relying on the periodic invalidation timer that runs on an interval based on the session timeout parameter, you can set specific times for the session management facility to scan for invalidated sessions in a distributed environment. When used with distributed sessions, this feature has the following benefits:
- You can schedule the scan for invalidated sessions for times of low application server activity, avoiding contention between invalidation scans of database or another WebSphere Application Server instance and read and write operations to service HTTP session requests.
- Significantly fewer external write operations can occur when running with the End of Service Method Write mode because the last access time of the session does not need to be written out on each HTTP request. (Manual Update options and Time Based Write options already minimize the writing of the last access time.)

Usage considerations
- With scheduled invalidation configured, HttpSession timeouts are not strictly enforced. Instead, all invalidation processing is handled at the configured invalidation times.
- HttpSessionBindingListener processing is handled at the configured invalidation times unless the HttpSession.invalidate method is explicitly called.

- The HttpSession.invalidate method immediately invalidates the session from both the session cache and the external store.
- The periodic invalidation thread still runs with scheduled invalidation. If the current hour of the day does not match one of the configured hours, sessions that have exceeded the invalidation interval are removed from cache, but not from the external store. Another request for that session results in returning that session back into the cache.
- When the periodic invalidation thread runs during one of the configured hours, all sessions that have exceeded the invalidation interval are invalidated by removal from both the cache and the external store.
- The periodic invalidation thread can run more than once during an hour and does not necessarily run exactly at the top of the hour.
- If you specify the interval for the periodic invalidation thread using the HttpSessionReaperPollInterval custom property, do not specify a value of more than 3600 seconds (1 hour) to ensure that invalidation processing happens at least once during each hour.

Configuring write contents

In session management, you can configure which session data is written to the database or to another WebSphere instance, depending on whether you are using database persistent sessions or memory to memory replication. This flexibility allows for fewer code changes for the JavaServer Pages (JSP) writer when the application will be operating in a clustered environment. The following options are available in Session Management for tuning what is to be written back:

- Write changed (the default) - Write only session data properties that have been updated through setAttribute method and removeAttribute method calls.
- Write all - Write all session data properties.

The **Write all** setting might benefit servlet and JSP writers who change Java objects' states that reside as attributes in HttpSession and do not call HttpSession.setAttribute method.

However, the use of **Write all** could result in more data being written back than is necessary. If this situation applies to you, consider combining the use of **Write all** with **Time-based write** to boost performance overall. As always, be sure to evaluate the advantages and disadvantages for your installation.

With either Write Contents setting, when a session is first created, complete session information is written, including all of the objects bound to the session. When using database session persistence, in subsequent session requests, what is written to the database depends on whether a single-row or multi-row schema has been set for the session database, as follows:

Write Contents setting	Behavior with single-row schema	Behavior with multirow schema
Write changed	If any session attribute is updated, all objects bound to the session are written.	Only the session data modified through setAttribute method or removeAttribute method calls is written.
Write all	All bound session attributes are written.	All session attributes that currently reside in the cache are written. If the session has never left the cache, all session attributes are written.

1. Go to the appropriate level of Session Management.
2. Click Distributed Environment Settings
3. Click Custom Tuning Parameters.
4. Select Custom Settings, and click Modify.
5. Select the appropriate write contents setting.

Configuring write frequency

In the Session Management facility, you can configure the frequency for writing session data to the database or to a WebSphere instance, depending on whether you use database distributed sessions or memory-to-memory replication. This flexibility enables you to weigh session performance gains against varying degrees of failover support. The following options are available in the Session Management facility for tuning write frequency:

- **End of service servlet**- Write session data at the end of the servlet service method call.
- **Manual update**- Write session data only when the servlet calls the IBMSession.sync method.
- **Time based** (the default) - Write session data at periodic intervals, in seconds (called the *write interval*).

When a session is first created, session information is always written at the end of the service call.

Using the time based write or manual update options can result in loss of data in failover scenarios since the backup copy of the session in the persistent store (for example, a database or another JVM) may not be in sync with the session in the session cache.

Base in-memory session pool size

The base in-memory session pool size number has different meanings, depending on session support configuration:

- With in-memory sessions, session access is optimized for up to this number of sessions.
- With distributed sessions (meaning, when sessions are stored in a database or in another WebSphere Application Server instance); it also specifies the cache size and the number of last access time updates saved in manual update mode.

For distributed sessions, when the session cache has reached its maximum size and a new session is requested, the Session Management facility removes the least recently used session from the cache to make room for the new one.

General memory requirements for the hardware system, and the usage characteristics of the e-business site, determines the optimum value.

Note that increasing the base in-memory session pool size can necessitate increasing the heap sizes of the Java processes for the corresponding WebSphere Application Servers.

Overflow in non-distributed sessions

By default, the number of sessions maintained in memory is specified by base in-memory session pool size. If you do not wish to place a limit on the number of sessions maintained in memory and allow overflow, set overflow to *true*.

Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site and creates sessions, but ignores any cookies or encoded URLs and never utilizes the same session from one HTTP request to the next.

When overflow is disallowed, the Session Management facility still returns a session with the HttpServletRequest getSession(true) method when the memory limit is reached, and this is an invalid session that is not saved.

With the WebSphere Application Server extension to HttpSession, com.ibm.websphere.servlet.session.IBMSession, an isOverflow method returns *true* if the session is such an invalid session. An application can check this status and react accordingly.

Controlling write operations

You can manually control when modified session data is written out to the database or to another WebSphere Application Server instance by using the sync method in the com.ibm.websphere.servlet.session.IBMSession interface. This interface extends the javax.servlet.http.HttpSession interface. By calling the sync method from the service method of a servlet, you send any changes in the session to the external location. When manual update is selected as the write frequency mode, session data changes are written to an external location only if the application calls the sync method. If the sync method is not called, session data changes are lost when a session object leaves the server cache. When end of service servlet or time based is the write frequency mode, the session data changes are written out whenever the sync method is called. If the sync method is not called, changes are written out at the end of service method or on a time interval basis based on the write frequency mode that is selected.

```
IBMSession iSession = (IBMSession) request.getSession();
iSession.setAttribute("name", "Bob");

//force write to external store
iSession.sync( )
```

If the database is down or is having difficulty connecting during an update to session values, the sync method always makes three attempts before it finally creates a BackedHashtable.getConnectionError error. For each connection attempt that fails, the BackedHashtable.StaleConnectionException is created and can be found in the sync method. If the database opens during any of these three attempts, the session data in the memory is then persisted and committed to the database.

However, if the database is still not up after the three attempts, then the session data in the memory is persisted only after the next check for session invalidation. Session invalidation is checked by a separate thread that is triggered every five minutes. The data in memory is consistent unless a request for session data is issued to the server between these events. For example, if the request for session data is issued within five minutes, then the previous persisted session data is sent.

Sessions are not transactional resources. Because the sync method is associated with a separate thread than the client, the exception that is created does not propagate to the client, which is running on the primary thread. Transactional integrity of data can be maintained through resources such as enterprise beans.

Tuning parameter settings

Use this page to set tuning parameters for distributed sessions.

To view this administrative console page, click **Servers > Application servers >***server_name***> Web container settings > Session management > Distributed environment settings >Custom tuning parameters**.

Tuning level:

Specifies that the session management facility provides certain predefined settings that affect performance.

Select one of these predefined settings or customize a setting. To customize a setting, select one of the predefined settings that comes closest to the setting desired, click **Custom settings**, make your changes, and then click **OK**.

Very high (optimize for performance)

Write frequency	Time based
Write interval	300 seconds
Write contents	Only updated attributes
Schedule sessions cleanup	true
First time of day default	0
Second time of day default	2

High

Write frequency	Time based
Write interval	300 seconds
Write contents	All session attributes
Schedule sessions cleanup	false

Medium

Write frequency	End of servlet service
Write contents	Only updated attributes
Schedule sessions cleanup	false

Low (optimize for failover)

Write frequency	End of servlet service
Write contents	All session attributes
Schedule sessions cleanup	false

Custom settings

Write frequency default	Time based
Write interval default	10 seconds
Write contents default	All session attributes
Schedule sessions cleanup default	false

Tuning parameter custom settings

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** > **Application servers** > *server_name***Web container settings** > **Session management** > **Distributed environment settings** > **Custom tuning parameters** > **Custom settings**.

Write frequency:

Specifies when the session is written to the persistent store.

End of servlet service	A session writes to a database or another WebSphere Application Server instance after the servlet completes execution.
Manual update	A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance.
Time based	Session data writes to the database or another WebSphere Application Server instance based on the specified Write interval value. Default: 10 seconds

Write contents:

Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

Only updated attributes	Only updated attributes are written to the persistent store.
All session attribute	All attributes are written to the persistent store.

Schedule sessions cleanup:

Specifies when to clean the invalid sessions from a database or another application server instance.

Specify distributed sessions cleanup schedule	Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions. When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance.

First Time of Day (0 - 23)	Indicates the first hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.
Second Time of Day (0 - 23)	Indicates the second hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled.

Web services

Monitoring the performance of Web services applications

This topic explains how to monitor the performance of a Web service that is implemented in the WebSphere Application Server.

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 that is described in the *Administering applications and their environment* PDF book. Select the Web module named webServicesModule in step 7.
2. Monitor performance with Tivoli Performance Monitor that is described in the *Administering applications and their environment* PDF book. In the left pane of the performance view, expand the host and server. Select **Web Services**. Run the Web services client application.

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 the PMI data organization information in the *Administering applications and their environment* PDF book.

If you are having problems with your Web services applications, read about problems and solutions in the *Administering applications and their environment* PDF book.

Web services performance best practices

Web services are developed and deployed based on standards provided by the Web Services for Java 2 Platform, Enterprise Edition (J2EE) specification, as 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 Service runtime, the SOAP message is translated based on the J2EE request. The J2EE-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 the monitoring the performance of Web services applications in the *Administering applications and their environment* PDF book for more information about analyzing and tuning Web services.

If you are using Web service application that was developed for a WebSphere Application Server version prior to Version 6.0, 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.

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 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 the *Administering applications and their environment* PDF book.
- Access to Web services over multiple transport protocols extends existing 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 the *Administering applications and their environment* PDF book.
- The SOAP with Attachments API for Java (SAAJ) 1.2 provides a low-level programming model for Web services relative to JAX-RPC. The SAAJ 1.2 API provides features to create and process SOAP requests using a low-level XML API. SAAJ 1.2 supports just-in-time parsing and other internal algorithms. For information about SAAJ 1.2 or Web services programming see the *Administering applications and their environment* PDF book.

- 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 1.2 implementation.

IBM provides considerable documentation and best practices for Web services application design and development that details these items and more. For a list of key Web sites that discuss performance best practices, see the *Administering applications and their environment* PDF book.

Data access resources

EJB Container Cache tuning

Monitoring Tivoli Performance Viewer (TPV) is a great way to diagnose if the EJB Container Cache size setting is tuned correctly for your application. If the application has filled the cache causing evictions to occur, TPV will show a very high rate of ejbStores() being called and probably a lower than expected CPU utilization on the application server machine.

All applications using enterprise beans should have this setting adjusted from the default if the following formula works out to more than 2000.

```
EJB_Cache_Size = (Largest number of Option B or C Entity Beans enlisted in a
transaction * maximum number of concurrent transactions) +
(Largest number of unique Option A Entity Beans expected to be accessed during
typical application workload) +
(Number of stateful Session Beans active during typical workload) +
(Number of stateless SessionBean types used during typical workload)

Where:
Option B and C Entity Beans are only held in the EJB cache during the lifetime
of the transaction they are enlisted in. Therefore, the first term in the formula
computes the average EJB cache requirements for these types of beans.

Option A Entity Beans are held in the EJB cache indefinitely, and are only removed
 from the cache if there start to become more beans in the cache than the cache
size has been set to.

Stateful Session Beans are held in the EJB cache until they are removed by the
application, or their session timeout value is reached.

Only a single stateless Session Bean instance for each EJB type is held in the
cache during the time any methods are being executed on that stateless Session
Bean. If two or more methods are being executed simultaneously on the same
stateless Session Bean type, each method executes on its own bean instance, but
only one cache location is used for all of these instances.
```

This calculates the upper bound on the maximum possible number of enterprise beans active at one time inside the application server. Because the EJB Containers cache is built to contain all these beans for performance optimizations, best performance can be achieved by setting this cache size to be larger than the number resulting from the calculation above.

```
<tuning parameter>
This setting can be found under Servers > Application Servers > serverName >
         EJB Container > EJB Cache Settings
```

Also while adjusting the EJB Cache Size, the EJB Container management thread parameter can be tuned to meet the needs of the application. The management thread is controlled through the Clean Up Interval setting. This setting controls how frequently a daemon thread inside of WebSphere Application Server wakes up and attempts to remove bean instances from the cache that have not been used recently, attempting to keep the number of bean instances at or below the cache size. This allows the EJB container to place and look up items in the cache as quickly as possible. It normally is best to leave this interval set to the default, however, in some cases, it may be worthwhile to see if there is a benefit to reducing this interval.

EJB Container Pool Size

If the application is using the majority of the instances in the pool, TPV indicates this. When this occurs, then the size of those bean pools that are being exhausted should be increased. This can be done by adding the following parameter in the JVM's custom properties tag .

```
-Dcom.ibm.websphere.ejbcontainer.poolSize=<application_name>#<module_name>#
<enterprisebean_name>=<minSize>,<maxSize>

where:
 <application_name> is the J2EE application name as defined in the application
archive (.ear) file deployment descriptor, for the bean whose pool size is being
set

<module_name> is the .jar file name of the EJB module, for the bean whose pool size
 is being set,

<bean_name> is the J2EE Enterprise Bean name as defined in the EJB module
deployment descriptor, for the bean whose pool size is being set

<minSize> is the number of bean instances the container maintains in the pool,
 irrespective of how long the beans have been in the pool (beans greater than this
 number are cleared from the pool over time to optimize memory usage)

<maxSize> is the number of bean instances in the pool where no more bean instances
 are placed in the pool after they are used (that is, once the pool is at this
 size, any additional beans are discarded rather than added into the pool --
this ensures the number of beans in the pool has an upper limit so memory usage
does not grow in an unbounded fashion).

To keep the number of instances in the pool at a fixed size, minSize and maxSize
can be set to the same number. Note that there is a separate instance pool for
every EJB type running in the application server, and that every pool starts out
with no instances in it - that is, the number of instances grows as beans are
used and then placed in the pool. When a bean instance is needed by the container
and no beans are available in the pool, the container creates a new bean
instance, uses it, then places that instance in the pool (unless there are
already maxSize instances in the pool).

For example, the statement
-Dcom.ibm.websphere.ejbcontainer.poolSize=ivtApp#ivtEJB.jar#ivtEJBObject=125,1327

would set a minSize of 125 and a maxSize of 1327 on the bean named "ivtEJBObject"
within the ivtEJB.jar file, in the application "ivtApp".
```

Where ivtApp is replaced by the actual application name, ivtEJB.jar is replaced by the jar containing the bean that needs to have its pool size increased, and ivtEJBObject is the bean name of the enterprise bean whose pool size should be increased. The 125,1327 is the minimum and maximum number of beans that will be held in the pool. These should be set so no more evictions occur from the pool and in most cases should be set equal if memory is plentiful because no growth and shrinkage of the pool will occur.

EJB Container Primary Key Mutation

Application developers and administrators should have a good idea of how their application handles the creation of primary key objects for use by container-managed persistence (CMP) beans and bean-managed persistence (BMP) beans inside of WebSphere Application Server. The IBM EJB Container uses the primary key of an Entity bean as an identifier inside of many internal data structures to optimize performance. However, the EJB Container must copy these primary key objects upon the first access to the bean to ensure that the objects stored in the internal caches are separate from the ones used in an application, in case the application changes or mutates the primary key, to keep the internal structures consistent.

If the application does not mutate any of the primary keys used to create and access entity beans after they are created, then a special flag can be used that allows the EJB Container to skip the copy of the primary key object, thus saving CPU cycles and increasing performance. This mechanism can be enabled *at your own risk* by adding the following –D property to the JVM custom property field.

```
<tuning parameter>
-Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation=true
```

The performance benefit of this optimization depends on the application. If the application uses primitive types for enterprise beans' primary keys there will be no gain because these objects are already immutable and the copy mechanism takes this into account. If, however, the application uses many complex primary keys (that is, And object for a primary key or multiple fields) then this parameter can yield significant improvements.

Persistence Manager Deferred Insert on EJB Create

The IBM Persistence manager is used by the EJB Container to persist data to the database from CMP entity beans. When creating entity beans by calling the ejbCreate() method, by default the Persistence manager immediately inserts the empty row with only the primary key in the database. In most cases applications, after creating the bean, modify fields in the bean created or in other beans inside of the same transaction. If the user wishes to postpone the insert into the database until the end of the transaction, so that it will eliminate one trip to the database, they may set this –D flag inside of the JVM custom properties field. The data will still be inserted into the database and consistency will be maintained.

```
<tuning parameter>
-Dcom.ibm.ws.pm.deferredcreate=true
```

The performance benefit of this optimization depends on the application. If the EJB applications transactions are very insert intensive the application could benefit largely from this optimization. If the application performs very few inserts then the benefit of this optimization will be much less.

Persistence Manager Database Batch Update on EJB Update

When an EJB application accesses multiple CMP beans inside of a single transaction, depending on the operations performed on the beans (updates, inserts, reads), the number of operations issued to the database will correspond directly to the operations performed on the CMP beans. If the database system you are using supports batching of update statements you can enable this flag and gain a performance boost on all interactions with the database that involve more than two updates in a single transaction. This flag will let the persistence manager add all the update statements into one single batch statement which will then be issued to the database. This saves round trips to the database, thus increasing performance. If the user knows their application exhibits the behavior of updating multiple CMP beans in a single transaction and the database supports batch updates they may set this –D flag inside of the JVM custom properties field.

```
<tuning parameter>
-Dcom.ibm.ws.pm.batch=true
```

The performance benefit of this optimization depends on the application. If the application never or infrequently updates CMP beans or only updates a single bean per transaction there will be no performance gain. If the application updates multiple beans per transaction then this parameter will benefit your applications performance.

DB2 tuning parameters

DB2 has many parameters that you can configure to optimize database performance. For complete DB2 tuning information, refer to the *DB2 UDB Administration Guide: Performance* document.

DB2 logging

- **Description:** DB2 has corresponding log files for each database that provides services to administrators, including viewing database access and the number of connections. For systems with multiple hard disk drives, you can gain large performance improvements by setting the log files for each database on a different hard drive from the database files.
- **How to view or set:** At a DB2 command prompt, issue the command: `db2 update db cfg for [database_name] using newlogpath [fully_qualified_path]`.
- **Default value:** Logs reside on the same disk as the database.
- **Recommended value:** Use a separate high-speed drive, preferably performance enhanced through a redundant array of independent disk (RAID) configuration.

DB2 configuration advisor

Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the database, and the database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

Number of connections to DB2 - MaxAppls and MaxAgents

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to establish clones, set the MaxAppls value as the maximum number of connections multiplied by the number of clones. The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources, set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

For example, MaxAppls = (number of connections set for the data source + number of connections in the session manager) multiplied by the number of clones.

After calculating the MaxAppls settings for the WebSphere Application Server database and each of the application databases, verify that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls values. For example, MaxAgents = sum of MaxAppls for all databases.

DB2 buffpage

- **Description:** Improves database system performance. Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. Data is accessed much faster from memory than from disk.
- **How to view or set:** To view the current value of buffpage for database *x*, issue the DB2 command `get db cfg for x` and look for the value **BUFFPAGE**. To set **BUFFPAGE** to a value of *n*, issue the DB2 command `update db cfg for x` using **BUFFPAGE *n*** and set **NPAGES** to -1 as follows:

  ```
  db2   <-- go to DB2 command mode, otherwise the following "select" does not work as is
      connect to x    <-- (where x is the particular DB2 database name)
      select * from syscat.bufferpools
         (and note the name of the default, perhaps: IBMDEFAULTBP)
         (if NPAGES is already -1, there is no need to issue following command)
      alter bufferpool IBMDEFAULTBP size -1
      (re-issue the above "select" and NPAGES now equals -1)
  ```

  You can collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio as follows:

  1. Collect the snapshot:
     a. Issue the **update monitor switches using bufferpool on** command.
     b. Make sure that bufferpool monitoring is on by issuing the **get monitor switches** command.
     c. Clear the monitor counters with the **reset monitor all** command.
  2. Run the application.
  3. Issue the **get snapshot for all databases** command before all applications disconnect from the database, otherwise statistics are lost.
  4. Issue the **update monitor switches using bufferpool off** command.
  5. Calculate the hit ratio by looking at the following database snapshot statistics:
     – Buffer pool data logical reads
     – Buffer pool data physical reads
     – Buffer pool index logical reads
     – Buffer pool index physical reads
- **Default value:** 250
- **Recommended value:** Continue increasing the value until the snapshot shows a satisfactory hit rate.

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk to service a page request. That is, the page is already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk input and output. Calculate the buffer pool hit ratio as follows:

- P = buffer pool data physical reads + buffer pool index physical reads
- L = buffer pool data logical reads + buffer pool index logical reads
- Hit ratio = (1-(P/L)) * 100%

DB2 query optimization level

- **Description:** Sets the amount of work and resources that DB2 puts into optimizing the access plan. When a database query runs in DB2, various methods are used to calculate the most efficient access plan. The range is from 0 to 9. An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.
- **How to view or set:** The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level that is specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the dft_queryopt setting. Dynamic SQL statements use the optimization class that is specified by the current query optimization special register, which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:

  ```
  Set current query optimization = 1
  ```

  If the current query optimization register is not set, dynamic statements are bound using the default query optimization class.
- **Default value:** 5
- **Recommended value:** Set the optimization level for the needs of the application. Use high levels only when there are very complicated queries.

DB2 reorgchk
- **Description:** Obtains the current statistics for data and rebinding. Use this parameter because SQL statement performance can deteriorate after many updates, deletes or inserts.
- **How to view or set:** Use the DB2 **reorgchk update statistics on table all** command to perform the **runstats** operation on all user and system tables for the database to which you are currently connected. Rebind packages using the **bind** command. If statistics are available, issue the **db2 -v ″select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes″** command on DB2 CLP. If no statistic updates exist, nleaf and nlevels are -1, and stats_time has an empty entry (for example: ″-″). If the runstats command was previously run, the real-time stamp from completion of the runstats operation also displays under stats_time. If you think the time shown for the previous runstats operation is too old, run the runstats command again.
- **Default value:** None
- **Recommended value:** None

DB2 locktimeout
- **Description:** Specifies the number of seconds that an application waits to obtain a lock. Setting this property helps avoid global deadlocks for applications.
- **How to view or set:** To view the current value of the lock timeout property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the value, LOCKTIMEOUT. To set LOCKTIMEOUT to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **LOCKTIMEOUT *n***, where *xxxxxx* is the name of the application database and *n* is a value between 0 and 30 000 inclusive.
- **Default value:** -1, meaning lock timeout detection is turned off. In this situation, an application waits for a lock if one is not available at the time of the request, until either of the following events occurs:
  - The lock is granted
  - A deadlock occurs
- **Recommended value:** If your database access pattern tends toward a majority of writes, set this value so that it gives you early warning when a timeout occurs. A setting of 30 seconds suits this purpose. If your pattern tends toward a

majority of reads, either accept the default lock timeout value, or set the property to a value greater than 30 seconds.

DB2 maxlocks

- **Description:** Specifies the percentage of the lock list that is reached when the database manager performs escalation, from row to table, for the locks held by the application. Although the escalation process does not take much time, locking entire tables versus individual rows decreases concurrency, and potentially decreases overall database performance for subsequent attempts to access the affected tables.
- **How to view or set:** To view the current value of the maxlocks property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the MAXLOCKS value. To set MAXLOCKS to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **MAXLOCKS *n***, where *xxxxxx* is the name of the application database and *n* is a value between 1 and 100 inclusive.
- **Default value:** Refer to the current database information for property default values per operating system.
- **Recommended value:** If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the locklist parameter, which is described in the following paragraph. You can use the database system monitor to determine if lock escalations are occurring.

DB2 locklist

- **Description:** Specifies the amount of storage that is allocated to the lock list.
- **How to view or set:** To view the current value of the locklist property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the LOCKLIST value . To set LOCKLIST to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **LOCKLIST *n***, where *xxxxxx* is the name of the application database and *n* is a value between 4 and 60 000 inclusive.
- **Default value:** Refer to the current database information for property default values per operating system.
- **Recommended value:** If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the maxlocks parameter, which is described in the previous paragraph. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *DB2 Administration Guide: Performance* document for more details.

Tuning parameters for data access resources

For better application performance, you can tune some data access resources through the WebSphere Application Server administrative console.

Tune these properties of data sources and connection pools to optimize the performance of transactions between your application and datastore.

Data source tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **WebSphere Application Server connection properties**.

Enable JMS one phase optimization support
If your application does not use JMS messaging, **do not** select this option. Activating this support enables the Java Message Service (JMS) to get optimized connections from the data source. Activating this support also *prevents* JDBC applications from obtaining connections from the data source. For further explanation of JMS one phase support, refer to the

article entitled ″Sharing connections to benefit from one phase commit optimization″ in this information center.

Statement cache size
Specifies the number of prepared statements that are cached per connection. (A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently run the given SQL statement multiple times.) In general, the more statements your application has, the larger the cache should be. **Be aware**, however, that specifying a larger statement cache size than needed wastes application memory and *does not* improve performance.

Determine the value for your cache size by adding the number of uniquely prepared statements, callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible prepared statements that are cached on a given connection over the life of the server. For additional information about this setting, see the *Administering applications and their environment* PDF book.

Default: For most databases the default is 10. Zero means there is no cache statement.

Connection pool tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **Connection pool settings**.

Maximum connections
Specifies the maximum number of physical connections that can be created in this pool. These are the physical connections to the backend datastore. When this number is reached, no new physical connections are created; requestors must wait until a physical connection that is currently in use is returned to the pool.

For optimal performance, set the value for the connection pool lower than the value for the Web container threadpool size. Lower settings, such as 10 to 30 connections, might perform better than higher settings, such as 100. For additional information about this setting, see the *Administering applications and their environment* PDF book.

Default: 10

Minimum connections
Specifies the minimum number of physical connections to maintain. Until this number is exceeded, the pool maintenance thread does not discard physical connections.

If you set this property for a higher number of connections than your application ultimately uses at run time, you do not waste application resources. WebSphere Application Server does not create additional connections to achieve your minimum setting. Of course, if your application requires more connections than the value you set for this property, application performance diminishes as connection requests wait for fulfillment. For additional information about this setting, see the *Administering applications and their environment* PDF book.

Default: 1

Database performance tuning

Database performance tuning can dramatically affect the throughput of your application. For example, if your application requires high concurrency (multiple, simultaneous interactions with backend data), an improperly tuned database can result in a bottleneck. Database access threads accumulate in a backlog when the database is not configured to accept a sufficient number of incoming requests.

Tuning parameters vary according to the type of database you are using. "DB2 tuning tips for z/OS" on page 31 are provided for your convenience. Because DB2 is not a WebSphere Application Server product and can change, consider these descriptions as suggestions.

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.

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 Version 5 queue destinations"
- "Performance considerations for WebSphere Version 5 topic destinations" on page 84
- "Performance considerations for WebSphere MQ queue destinations" on page 84
- "Performance considerations for WebSphere MQ topic destinations" on page 85

Performance considerations for WebSphere 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 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 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.

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

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 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.sh` process is running, not the user ID specified in the wsadmin request.

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 specified in soap.client.props
- `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 WAS CA Certificates to Servers keyring                      */
   "RACDCERT ID(yyyyy) CONNECT (RING(xxxxxx) LABEL('zzzzzzz')   CERTAUTH"
   SETROPTS RACLIST(FACILITY) REFRESH"
   ```

Setting up SSL connections for Java clients

To configure SSL for use between Java clients running on a workstation and the WebSphere Application Server for z/OS Java 2 Platform, Enterprise Edition (J2EE) 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: `c:\tmp` directory
5. Add the digital certificate to the TrustStore used by the client. For example: DummyClientTrustFile.jks file: `keytool -import -file c:\tmp\IBMUSER.WAS.CA -keystore DummyClientTrustFile.jks]`

Tuning security configurations

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing 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 must disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your J2EE roles? These questions are things to consider when designing your security infrastructure.

Complete the following steps for general security tuning:

1. 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.
2. Disable security for the specific application server that does not require resource protection because some application servers do not have protected resources. If the application server needs to go downstream with credentials,

however, this action might not be feasible. There are two ways to disable security. Use either of the following two procedures and then restart the application server:

Using the administrative console

a. Click **Security > Global security** .
b. Disable the **Enable global security** option.

Using the command line

a. Type `wsadmin.sh -conntype NONE`.
b. When the system command prompt redisplays, type `securityoff`.

3. 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 global security in a Network Deployment environment.

   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 nodes.

   If the cell is in a configuration state (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.

   Refer to the article, Enabling global security in the WebSphere Application Server Network Deployment package for more detailed information about enabling security in a distributed environment.
4. Consider increasing the cache and token time-out if you feel your environment is secure enough. By doing so, 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 time-out 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.
5. 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.
6. 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.
7. Consider tuning the Object Request Broker (ORB) because it is a factor in enterprise bean performance with or without security enabled. Refer to the article, .
8. If using SSL, then enable the SSL session tracking mechanism option as described in the ″Session management settings″ article in the *Administering applications and their environment* PDF book

Tuning CSIv2

1. Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Since you are already making the

SSL connection, using mutual authentication adds little overhead while removing the service context containing the user ID and password completely.

2. 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.
3. Consider putting just 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, we simply trust the identity token. The SSL connection trusts the server by way of client certificate authentication.
4. Ensure that stateful sessions are enabled for Common Secure Interoperability Version 2 (CSIv2). This is the default, but only requires authentication on the first request and any subsequent token expirations.
5. If you are only communicating with WebSphere Application Server Version 6 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.

Tuning LDAP authentication

1. Select the **Ignore case for authorization** option in the WebSphere Application Server Lightweight Directory Access Protocol (LDAP) User Registry configuration, when case-sensitivity is not important.
2. Select the **Reuse connections** option in the WebSphere Application Server LDAP User Registry configuration.
3. Check to see which caches your LDAP server has and take advantage of them. This action is best with LDAP servers that do not change frequently.
4. Choose the directory type of either IBM Tivoli Directory Server or SecureWay, 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, it is required that authorization is case insensitive to use IBM Tivoli Directory Server.
5. 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 increases performance in group membership lookup. However, only use **Role** for group mechanisms.

Tuning Web authentication

1. Consider increasing the cache and token time-out 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 already created. The downside of increasing the token time-out is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.
2. Consider enabling single signon (SSO). SSO is only available when you select **LTPA** as the authentication mechanism in the Global security panel. 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. There are some situations where SSO is not desirable and should not be used in those situations.

3. Consider disabling or enabling the **Web Inbound Security Attribute Propagation** option on the SSO panel if the function is not required. In some cases, having the function enabled improves 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 improves performance. This improvement is most likely when the user registry calls do not take considerable resources.

Tuning authorization

1. Consider mapping your users to groups in the user registry. Then associate the groups with your Java 2 Platform, Enterprise Edition (J2EE) roles. This association greatly improves performance as the number of users increases.
2. Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all 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 required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
3. Judiciously assign security-constraints for servlets. For example, you can use the URL pattern `*.jsp` to apply the same authentication data constraints to indicate all JSP files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the extension match (`*.jsp`, `*.do`, `*.html`) if there is no exact match and longest path match for a given URL in the security constraints.

There is always 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, you should create a benchmark before making any change to ensure the change is improving performance.

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 configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

Security cache properties

The following security cache custom properties determine whether the authentication cache is enabled or disabled. If the authentication cache is enabled, as recommended, these custom properties specify the initial size of the primary and secondary hash table caches, which affect the frequency of rehashing and the distribution of the hash algorithms. To specify these security cache properties, complete the following steps:

1. Click **Servers > Application Servers >** *server_name* .
2. Under Server infrastructure, click **Java and Process Management > Process definition > Control** or **Servant**.
3. Under Additional properties, click **Java Virtual Machine > Custom Properties**
4. Click **New** to specify a new custom property.

Important: The com.ibm.websphere.security.util.tokenCacheSize and com.ibm.websphere.security.util.LTPAValidationCacheSize properties were replaced with the com.ibm.websphere.security.util.authCacheSize property.

WebSphere Application Server includes the following security cache custom properties:

com.ibm.websphere.security.util.authCacheEnabled
Specifies whether to disable the authentication cache. It is recommended that you leave the authentication cache enabled for performance reasons. However, you might disable the authentication cache for debug or measurement purposes.

Default:	True

com.ibm.websphere.security.util.authCacheSize
Specifies the initial size of the primary and secondary hash table caches. A higher number of available hash values might decrease the occurrence of hash collisions. A hash collision results in a linear search for the hash bucket, which might decrease the retrieval time. If several entries compose a hash table cache, you can might consider creating a table with a larger capacity that supports more efficient hash entries instead of allowing automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

Default:	200
Type:	Integer

Tuning security

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

While it is not possible 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 for z/OS” on page 45.

- Fine-tune the **Cache timeout** value on the Global security panel in the administrative console. For more information, see the *Securing applications and their environment* PDF book.
- Configure the security cache properties. For more information, see *Securing applications and their environment* PDF book.
- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console. For more information, see the *Administering applications and their environment* PDF book.
- Modify the RACF security settings as documented in the “RACF tuning tips for z/OS” on page 35 article.
- Read the “Tuning security configurations” on page 86 article for more information.

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 Learn about WebSphere applications: Overview and new features for an introduction to each WebSphere extension.

ActivitySessions	How do I?...	Overview	Samples	
Application profiling	How do I?...	Overview	Samples	
Asynchronous beans	How do I?...	Overview	Samples	
Dynamic caching	How do I?...	Overview		
Dynamic query	How do I?...	Overview	Samples	
Internationalization	How do I?...	Overview	Samples	
Object pools	How do I?...	Overview		
Scheduler	How do I?...	Overview	Samples	
Startup beans	How do I?...	Overview		
Work areas	How do I?...	Overview		

Dynamic cache

Managing cache entries stored on a disk

Use this page to set Java virtual machine (JVM) custom properties to maintain cache entries that are saved to disk.

Steps for this task

You can set the custom properties globally to affect all cache instances, or you can set the custom property on a single cache instance. In most cases, set the properties on the individual cache instances. To set the custom properties on the default cache instance, use the global option. If you set the same property both globally and on a cache instance, the value that is set on the cache instance overrides the global value.

To configure the custom properties on a single object cache instance or servlet cache instance, perform the following steps:

1. In the administrative console, click one of the following paths:
   - To configure a servlet cache instance, click **Resources > Cache instances > Servlet cache instances >** *servlet_cache_instance_name* **> Custom properties > New**.
   - To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name* **> Custom properties > New**.
2. Type the name of the custom property. When configuring these custom properties on a single cache instance, you do not use the full property path. For example, type `htodCleanupFrequency` to configure the com.ibm.ws.cache.CacheConfig.htodCleanupFrequency custom property.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

To configure the custom property globally across all configured cache instances, perform the following steps:

1. In the administrative console, click **Servers > Application servers >** *server_name* **> Java and process management > Process definition > Java virtual machine > Custom properties > New**.

2. Type the name of the custom property (for example, com.ibm.ws.cache.CacheConfig.htodCleanupFrequency) in the **Name** field.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

com.ibm.ws.cache.CacheConfig.htodCleanupFrequency

Use this property to change the amount of time between disk cache cleanup.

By default, the disk cache cleanup is scheduled to run at midnight to remove expired cache entries and cache entries that have not been accessed in the past 24 hours. However, if you have thousands of cache entries that might expire within one or two hours, the files that are in the disk cache can grow large and become unmanageable. Use the com.ibm.ws.cache.CacheConfig.htodCleanupFrequency custom property to change the time interval between disk cache cleanup.

Units	minutes For example, a value of 60 means 60 minutes between each disk cache cleanup.
Default	0 The disk cache cleanup occurs at midnight every 24 hours.

Tune the delay offload function

Use these properties to tune the delay offload function for the disk cache. The delay offload function uses extra memory buffers for dependency IDs and templates to delay the disk offload and minimize the input and output operations. However, if most of your cache IDs are longer than 100 bytes, the delay offload function might use too much memory. Use any combination of the following properties to tune your configuration:

- To increase or decrease the in-memory limit of cache IDs for dependency ID and template buffers, use the com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit custom property.
- To disable the disk cache delay offload function, use the com.ibm.ws.cache.CacheConfig.htodDelayOffload custom property. Disabling this property saves all cache entries to disk immediately after removing them from the memory cache.

com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit

Use this property to specify the number of different cache IDs that can be saved in memory for the dependency ID and template buffers. Consider increasing this value if you have a lot of memory in your server and you want to increase the performance of your disk cache.

Units	number of cache IDs For example, a value of 1000 means that each dependency ID or template ID can have up to 1000 different cache IDs in memory.
Default	1000

Minimum	100

com.ibm.ws.cache.CacheConfig.htodDelayOffload

Use this property to specify if extra memory buffers should be used in memory for dependency IDs and templates to delay disk offload and to minimize input and output operations to the disk. This property is enabled by default. However, consider disabling this property if your cache IDs are larger than 100 bytes because this option might use too much memory when it buffers your data. If you set this property to `false`, all the cache entries are copied to disk immediately after they are removed from the memory cache.

Default	true

Tuning dynamic cache with the cache monitor

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

Verify that dynamic cache is enabled and that the cache monitor application is installed on your application server. See the *Setting up the application serving environment* PDF book 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.

Cache configuration value	Description
Servlet Caching Enabled	If servlet caching is enabled, results from servlets and JavaServer Pages (JSP) files are cached. See the *Setting up the application serving environment* PDF book for more information.
Disk Offload Enabled	Specifies if entries that are being removed from the cache are saved to disk. See the *Setting up the application serving environment* PDF book 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 *Setting up the application serving environment* PDF book to increase the cache size.

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

See the *Setting up the application serving environment* PDF book for more information about the dynamic cache.

Work area

Work area service performance considerations

The work area service is designed to address complex data passing patterns that can quickly grow beyond convenient maintenance. A *work area* is a note pad that is accessible to any client that is capable of looking up Java Naming Directory Interface (JNDI). After a work area is established, data can be placed there for future use in any subsequent method calls to both remote and local resources.

You can utilize a work area when a large number of methods require common information or if information is only needed by a method that is significantly further down the call graph. The former avoids the need for complex parameter passing models where the number of arguments passed becomes excessive and hard to maintain. You can improve application function by placing the information in a work area and subsequently accessing it independently in each method, eliminating the need to pass these parameters from method to method. The latter case also avoids unnecessary parameter passing and helps to improve performance by reducing the cost of marshalling and de-marshalling these parameters over the Object Request Broker (ORB) when they are only needed occasionally throughout the call graph.

When attempting to maximize performance by using a work area, cache the UserWorkArea partition that is retrieved from JNDI wherever it is accessed. You can reduce the time spent looking up information in JNDI by retrieving it once and keeping a reference for the future. JNDI lookup takes time and can be costly.

Additional caching mechanisms available to a user-defined partition are defined by the configuration property, ″Deferred Attribute Serialization″. This mechanism attempts to minimize the number of serialization and deserialization calls.

The maxSendSize and maxReceiveSize configuration parameters can affect the performance of the work area. Setting these two values to 0 (zero) effectively turns off the policing of the size of context that can be sent in a work area. This action can enhance performance, depending on the number of nested work areas an application uses. In applications that use only one work area, the performance enhancement might be negligible. In applications that have a large number of nested work areas, there might be a performance enhancement. However, a user must note that by turning off this policing it is possible that an extremely large amount of data might be sent to a server.

Performance is degraded if you use a work area as a direct replacement to passing a single parameter over a single method call. The reason is that you incur more overhead than just passing that parameter between method calls. Although the degradation is usually within acceptable tolerances and scales similarly to passing parameters with regard to object size, consider degradation a potential problem before utilizing the service. As with most functional services, intelligent use of the work areas yields the best results.

The work area service is a tool to simplify the job of passing information from resource to resource, and in some cases can improve performance by reducing the overhead that is associated with a parameter passing when the information is only sparsely accessed within the call graph. Caching the instance retrieved from JNDI is important to effectively maximize performance during runtime.

Chapter 8. Troubleshooting performance

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

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 Performance Monitoring Infrastructure (PMI), which is described in the *Administering applications and their environment* PDF book, 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, which is described in the *Administering applications and their environment* PDF book, 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, which is described in the *Administering applications and their environment* PDF book, to check for resource consumption, including frequency of garbage collections. You might need code profiling tools to isolate the problem to a specific method.
- **Scenario:** Poor performance only occurs with multiple users.

  Suggested solution: Check to determine if any systems have high CPU, network or disk utilization and address those. For clustered configurations, check for uneven loading across cluster members.
- **Scenario:** None of the systems seems to have a CPU, memory, network, or disk constraint but performance problems occur with multiple users.

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

Eliminating a bottleneck

Consider the following methods to eliminate a bottleneck:

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

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

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

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

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

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

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

Additional references

IBM WebSphere V6: Performance, Scalability, and High Availability Concepts

WebSphere Application Server Performance Web site

All SPEC jAppServer2004 Results Published by SPEC.

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

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).