WebSphere® Application Server—Express for i5/OS, Version 6.1

Tuning guide

Note

Before using this information, be sure to read the general information under “Notices” on page 121.

Compilation date: May 5, 2006

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

Use the links provided in this topic to learn about tuning applications and their environment.

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

Tuning parameter hot list

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

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

- **Review the hardware and software requirements**

  Review the hardware and software requirements on the IBM WebSphere Application Server supported hardware, software, and APIs Web site to get started.
- **Install the latest WebSphere Application Server PTFs**

  The list of recommended updates is maintained on the Business solutions
- **Check hardware configuration and settings**

  Check network connections to make sure that they are running at their highest speed. For more information, see Chapter 6, “Tuning the application serving environment,” on page 25.
- **Review your application design**

  You can track many performance problems back to the application design. Review the design to determine if it causes performance problems.
- **“Tuning i5/OS systems” on page 35**

  Operating system configuration plays a key role in performance. In many cases, adjustments to some TCP/IP parameters might be necessary for your application.
- **Set the minimum and maximum Java virtual machine (JVM) heap sizes**

  Many applications need a larger heap size for best performance.
- **Use a type 2 JDBC driver for local data access and a type 4 (or pure Java) JDBC driver for remote data access**

  In general, the type 2 JDBC driver is recommended. Use the link above to view a list of database vendor-specific requirements, which can tell you if a type 4 JDBC driver is supported for your database.

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

  Use applications that can take advantage of the pass by reference option to avoid the cost of copying parameters to the stack.
- **Tune related components, for example, database**

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

  For more information, see the *Administering applications and their environment* PDF for more information.

Chapter 2. How do I tune performance?

Hold your cursor over the task icon () to see a description of the task. The task preview feature is unavailable for Mozilla Web browsers.

Tune the application serving environment Related documentation topics: • Performance Capabilities Reference for iSeries • WebSphere Application Server Performance Tuning Resources for iSeries	Documentation
Obtain tuning advice: Performance advisor	Documentation
Obtain tuning advice: TPV advisor	Documentation
Tune WebSphere applications	Documentation
Pool database connections	Documentation
Configure caching	Documentation

Legend for ″How do I?...″ links

Detailed steps	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+

Chapter 3. Planning for performance

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

- Consult the following Web resources for learning.

  IBM Patterns for e-Business

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

  Planning for availability in the enterprise

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

  Hardware configurations for WebSphere Application Server production environments
  This article describes the most common production hardware configurations, and provides the reasons for choosing each one. It begins with a single machine configuration, and then proceeds with additional configurations that have higher fault tolerance, horizontal scaling, and a separation of Web and enterprise bean servers.
- See the documentation for the product functionality to improve performance .

Application design consideration

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

Consult the Designing applications topic in the *Developing and deploying applications* PDF, which highlights Web sites and other ideas for finding best practices for designing WebSphere applications, particularly in the realm of WebSphere extensions to the Java 2 Platform, Enterprise Edition (J2EE) specification.

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

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

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

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 solution that has clean architecture. The clean architecture considers data consistency, security, maintenance, portability, and the performance of that solution. Although this approach might not yield the absolute peak performance obtainable from manual coding a solution that ignores the mentioned qualities of service, this approach can achieve the appropriate balance of data consistency, maintainability, portability, security, and performance.

Multiple options are available in 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 of the Tuning application servers topic on tuning enterprise beans and more specifically, CMP.

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

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

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

Model-view-controller pattern

One of the standard 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:
- “Session management tuning” on page 37
- “EJB Container tuning” on page 43
- “Tuning dynamic cache with the cache monitor” on page 102

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.

  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)

  – Designing applications.

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

Chapter 5. Obtaining advice from the advisors

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

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

Tuning WebSphere Application Server is a critical part of getting the best performance from your Web site. However, tuning WebSphere Application Server involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data, and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.

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

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

Additionally, you can use the heap monitor feature to monitor the Java Virtual Machine (JVM) heap size of a WebSphere Application Server profile in comparison to pool size. The feature is available for new WebSphere Application Server profiles or profiles that are created after you update to the WebSphere Application Server. For existing WebSphere Application Server profiles, there is a script available to add the feature. See The heapMonitor script for more information.

Why you want to use the performance advisors

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

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

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

For example, consider the data source statement cache. It optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not used in an active connection. (Both statements are SQL statements that essentially run repeatable tasks without the costs of repeated

compilation.) If the cache is full, an old entry in the cache is discarded to make room for the new one. The best performance is generally obtained when the cache is large enough to hold all of the statements that are used in the application. The PMI counter, `prepared statement cache discards`, indicates the number of statements that are discarded from the cache. The performance advisors check this counter and provide recommendations to minimize the cache discards.

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.

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

Performance advisor types and purposes

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

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

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

Types of advice	Performance advice: • Object Request Broker (ORB) service thread pools • Web container thread pools • Connection pool size • Persisted session size and time • Prepared statement cache size • Session cache size • Memory leak detection Diagnostic advice: • Connection factory diagnostics • Data source diagnostic	Performance advice: • ORB service thread pools • Web container thread pools • Connection pool size • Persisted session size and time • Prepared statement cache size • Session cache size • Dynamic cache size • Java virtual machine (JVM) heap size • DB2 Performance Configuration wizard

Performance and Diagnostic Advisor

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

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

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

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

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

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

Read “Using the Performance and Diagnostic Advisor” on page 15 for startup and configuration steps.

Diagnostic alerts

In WebSphere Application Server Version 6.1 the Performance and Diagnostic Advisors are extended to provide more diagnostic alerts to help common troubleshoot problems.

Several alerts are made available to monitor connection factory and data sources behavior. See the *Administering applications and their environment* PDF for more information. Some of these alerts are straightforward and easy to comprehend. Others are much more involved and are intended for use by IBM support only.

ConnectionErrorOccured diagnostic alert

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

Connection low-percent efficiency diagnostic alert

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

Pool low-percent efficiency diagnostic alert

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

Surge mode entered or exited diagnostic alert

When surge mode is configured, an alert is sent whenever surge mode engages or disengages. See the surge mode documentation in the *Administering applications and their environment* PDF for more information.

Stuck connection block mode entered or exited diagnostic alert

When stuck connection detection is configured, an alert is sent whenever stuck connection blocking starts or stops. See the stuck connection documentation in the *Administering applications and their environment* PDF .

Local transaction containment (LTC) nesting threshold exceeded diagnostic alert

For LTC definition, see the Local transaction containment (LTC) and Transaction type and connection behavior topics in the *Administering applications and their environment* PDF, and Default behavior of managed connections in WebSphere Application Server topic.

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

Thread maximum connections exceeded diagnostic alert

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

Serial reuse violation diagnostic alert

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

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

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 Performance and Diagnostic Advisor

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

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

    This action also resets the state of the Performance and Diagnostic Advisor. For example, the current warning count is reset to zero (0) for each message.

13. Simulate a production level load. If you use the Performance and Diagnostic Advisor in a test environment, do any other tuning for performance, or simulate a realistic production load for your application. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory, to the levels that are expected in production. The Performance and Diagnostic Advisor provides advice when CPU utilization exceeds a sufficiently high level only. For a list of IBM business partners that provide tools to drive this type of load, see the topic, Performance: Resources for learning in the subsection of Monitoring performance with third-party tools.
14. Select the check box to enable the Performance and Diagnostic Advisor.

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

    profile_root/logs/*server_name*

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

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

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

You can enable and disable advice in the Advice Configuration panel. Some advice applies only to certain configurations, and can be enabled only for those configurations. For example, unbounded Object Request Broker (ORB) service thread pool advice is only relevant when the ORB service thread pool is unbounded, and can only be enabled when the ORB thread pool is unbounded. For more information on Advice configuration, see the topic, “Advice configuration settings” on page 18.

Performance and Diagnostic Advisor configuration settings

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

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

Enable Performance and Diagnostic Advisor Framework

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

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

- ThreadPools (module)

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

Enable automatic heap dump collection

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

Calculation Interval

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

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

Maximum warning sequence

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

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

Number of processors

Specifies the number of processors on the server.

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

Minimum CPU For Working System

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

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

The Minimum CPU for working system sets a lower limit as to when you should consider adjusting thread pools. For example, say you set this value to 50%. If the CPU is less than 50%, concurrency rules *do not*

try to free up resources by decreasing pools to get rid of unused threads. That is, if the pool size is 50-100 and only 20 threads are consistently used then concurrency rules would like to decrease the minimum pool size to 20.

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

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

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

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

To view this administrative page, click **Servers** > **Application Servers** > *server_name* . Under the Performance section, click **Performance and Diagnostic Advisor Configuration** > **Performance and Diagnostic Advice Configuration**.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Additional explanatory data follows.

Average number of threads: 48.

Configured maximum pool size: 2.

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

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

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

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

1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* **> Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Enable the Performance and Diagnostic Advisor Framework.
5. Click **OK**.

6. From the Runtime or Configuration tab of Performance and Diagnostic Advisor Framework, click **Performance and Diagnostic Advice configuration**.
7. Start the memory leak detection advice and stop any other unwanted advice.

The memory leak detection advice is started.

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

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the "Viewing the Performance and Diagnostic Advisor recommendations" on page 19 topic.

Lightweight memory leak detection

This topic describes memory leaks in Java applications and introduces lightweight memory leak detection, a new function available in WebSphere Application Server Version 6.0.2 and above.

Memory leaks in Java applications

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

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

Lightweight memory leak detection in WebSphere Application Server

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

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

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

Generating and analyzing heap dump

Use this task to generate and analyze heap dump on i5 operating systems.

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

To help you analyze memory leak problems when memory leak detection occurs, use the Heap Analysis Tools for Java™. Use the Heap Analysis Tools component (also known as Heap Analyzer) to perform Java application heap analysis and object create profiling (size and identification) over time. Heap Analyzer includes information about:

- Java virtual machine (JVM) heap growth or size
- The objects being created that include type of object, count and object size, object heap size
- The application ″Heap Footprint″ for memory sizing and performance considerations
- Includes a call stack for every snapshot when running in profile mode so objects created can be correlated to functions in the application.

The Heap Analyzer tool is a component of the iDoctor for iSeries suite of performance monitoring tools.

Use the heap monitor feature to monitor the JVM heap size of a WebSphere Application Server profile in comparison to pool size.

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 “Performance and Diagnostic Advisor” on page 13. 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 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.
2. In the administrative console, click **Monitoring and Tuning** > **Performance Viewer** > **Current activity**.
3. Simulate a production level load. Simulate a realistic production load for your application, if you use the performance advisor in a test environment, or do any other performance tuning. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory to the levels that are expected in production. The performance advisor only provides advice when CPU utilization exceeds a sufficiently high level. For a list of IBM business partners providing tools to drive this type of load, see the article, Performance: Resources for learning in the subsection of Monitoring performance with third party tools.
4. Log performance data with TPV.
5. Clicking **Refresh** on top of the table of advice causes the advisor to recalculate the advice based on the current data in the buffer.
6. Tuning advice displays when the Advisor icon is chosen in the TPV Performance Advisor. Double-click an individual message for details. Because PMI data is taken over an interval of time and averaged to provide advice, details within the advice message display as averages.

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

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

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

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

For more information about configuring user and logging settings of TPV, refer to the Configuring TPV settings article.

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

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

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

Performance advisor report in Tivoli Performance Viewer

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

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

Message

Specifies recommendations for performance tuning.

Click the message to obtain more details.

Performance data in the upper panel

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

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

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

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

Heap monitor

You can use the heap monitor feature to monitor the Java Virtual Machine (JVM) heap size of a WebSphere Application Server profile in comparison to pool size.

The feature is available for new WebSphere Application Server profiles or profiles that are created.

For existing WebSphere Application Server profiles, there is a script available to add the feature. See the *Using the administrative clients* PDF for more information.

Heap monitor default operation

The heap monitor follows default operation behavior as described in this file.

An active heap monitor typically sends a message to the QSYSOPR message queue when the WebSphere Application Server profile starts. For example, the Display Message command (DSPMSG QSYSOPR) displays the following message:

```
HEAP MONITOR STARTED FOR 012500/QEJBSVR/SERVER1 IN SUBSYSTEM QWAS61 IN POOL
  *BASE POOL ID=2 POOLSIZE(B)=1687994368 RESERVED(B)=778240 HEAP
  TOTAL(B)=202276864 FREE(B)=67037600 USEDHEAP=135239264
  OS400.GC.HEAP.SIZE.MAX(KB)  =240000000
```

In default operation, a similar message displays ENDED instead of STARTED when the WebSphere Application Server profile is ended. For example:

```
HEAP MONITOR ENDED FOR 012500/QEJBSVR/SERVER1 IN SUBSYSTEM
 QWAS61 IN POOL *BASE POOL ID=2 POOLSIZE(B)=6662139904 RESERVED(B)=5165056
 HEAP TOTAL(B)=312999936 FREE(B)=168637264 USEDHEAP=144362672
 OS400.GC.HEAP.SIZE.MAX(KB)  =240000000
```

The Display Log command (DSPLOG LOG(QHST) MSGID(CPI8859)) shows all STARTED and ENDED messages in the history log.

The default operation monitors the size of the Java Virtual Machine (JVM) Garbage Collection (GC) heap against the following:

- The size of the effective memory pool.
- The size of the memory pool size minus the reserved size.

It also issues a message if the effective memory pool size exceeds 85, 90, 95, or 100 percent. For example:

```
048241/QEJBSVR/SERVER1 GC HEAP USES 95% OF THE NON-RESERVED POOL.  JVM GC
   HEAP SIZE(KB)  EFFECTIVE POOLSIZE(KB):840282 882444.
 048241/QEJBSVR/SERVER1 GC HEAP USES 110% OF THE NON-RESERVED POOL.  JVM GC
   HEAP SIZE(KB)  EFFECTIVE POOLSIZE(KB):974601 882392.
```

The first number is the size of the heap, such as 840282 or 974601. The second number is the effective pool size (or non-reserved pool size), such as 882444 or 882392. The Display Log command (DSPLOG LOG(QHST) MSGID(CPF9898)) shows warning messages in the history log.

For the maximum Garbage Collection heap size, the default operation is to monitor the size of the JVM Garbage Collection heap so that it does not exceed 85, 90, or 95 percent of the maximum. For example:

```
048358/QEJBSVR/USER JAVA USED 88% OF THE GC HEAP.  USED HEAP SIZE(KB)
  AND MAX HEAP(KB):909088 1024001.
```

The maximum heap size is 1024001 Kbytes (-Xmx1000m), and the used heap size is 909088 Kbytes.

Activating the heap monitor

This task describes the steps used to activate the heap monitor. Heap monitor is used with WebSphere Application Server profiles to monitor heap size of a profile in comparison to pool size.

For existing WebSphere Application Server profiles, there is a script available to add the feature. See the *Using the administrative clients* PDF for more information.

To check if a WebSphere Application Server profile has the heap monitor enabled and to activate it if necessary, perform the following steps.

1. Start the server for the WebSphere Application Server profile.
2. Run the heapMonitor script with the -status flag. For example, for a WebSphere Application Server version 6.1 profile named default, enter the following command in the Qshell environment:

   ```
   /QIBM/ProdData/WebSphere/AppServer/V61/Base/bin/heapMonitor -profileName default -status
   ```

   The output should look similar to the following:

   ```
   WASX7209I: Connected to process "server1" on node MYSERVER using SOAP connector;
    The type of process is: UnManagedProcess
    WASX7303I: The following options are passed to the scripting
     environment and are available as argument that is stored in the argv
     variable: "[status, server1]"
    HEAP0002I: The heap monitor is disabled.
    $
   ```

3. To enable the heap monitor for this example, enter the following command in the Qshell environment:

   ```
   /QIBM/ProdData/WebSphere/AppServer/V61/Base/bin/heapMonitor -profileName default -enable
   ```

   The output should look similar to the following:

   ```
   WASX7209I: Connected to process "server1" on node MYSERVER using SOAP connector;
    The type of process is: UnManagedProcess
    WASX7303I: The following options are passed to the scripting environment and are
     available as argument that is stored in the argv
     variable: "[enable, server1]"
    HEAP0005I: Enabling the heap monitor...
    HEAP0003I: The heap monitor has been enabled.
    $
   ```

4. Stop and start the server.

The following message typically appears in the Display Message command (DSPMSG QSYSOPR):

```
HEAP MONITOR STARTED FOR 012500/QEJBSVR/SERVER1 IN SUBSYSTEM QWAS61 IN POOL
  *BASE POOL ID=2 POOLSIZE(B)=1687994368 RESERVED(B)=778240 HEAP
  TOTAL(B)=202276864 FREE(B)=67037600 USEDHEAP=135239264
  OS400.GC.HEAP.SIZE.MAX(KB)  =240000000
```

The heap monitor is activated.

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 runtime 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.

The tuning guide focuses on server tuning. If you want to tune your applications, see the Performance: Resources for learning article for more information about application tuning.

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.

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

Tuning parameter hot list

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

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

- **Review the hardware and software requirements**

  Review the hardware and software requirements on the IBM WebSphere Application Server supported hardware, software, and APIs Web site to get started.
- **Install the latest WebSphere Application Server PTFs**

  The list of recommended updates is maintained on the Business solutions
- **Check hardware configuration and settings**

  Check network connections to make sure that they are running at their highest speed. For more information, see Chapter 6, “Tuning the application serving environment.”
- **Review your application design**

  You can track many performance problems back to the application design. Review the design to determine if it causes performance problems.
- **“Tuning i5/OS systems” on page 35**

Operating system configuration plays a key role in performance. In many cases, adjustments to some TCP/IP parameters might be necessary for your application.

- **Set the minimum and maximum Java virtual machine (JVM) heap sizes**

  Many applications need a larger heap size for best performance.
- **Use a type 2 JDBC driver for local data access and a type 4 (or pure Java) JDBC driver for remote data access**

  In general, the type 2 JDBC driver is recommended. Use the link above to view a list of database vendor-specific requirements, which can tell you if a type 4 JDBC driver is supported for your database.

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

  Use applications that can take advantage of the pass by reference option to avoid the cost of copying parameters to the stack.
- **Tune related components, for example, database**

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

  For more information, see the *Administering applications and their environment* PDF for more information.

Tuning TCP/IP buffer sizes

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

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

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

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

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

Some system-wide settings can override the default 8 KB buffer size for sockets. With some applications, for exmaple, WebSphere Commerce Suite, a buffer size of 180 KB reduces flow control and typically does

not adversely affect paging. The optimal value is dependent on specific system characteristics. You might need to try several values before you determine the ideal buffer size for your system. To change the system wide value, perform the following steps:

- Change the TCP/IP configuration.
  1. Run the Change TCP/IP Attribute, **CHGTCPA** command.
  2. View and change the buffer sizes. On the Change TCP/IP Attributes window, press **F4**. The buffer sizes are displayed as the TCP receive and send buffer sizes. Type new values and save your changes.
- Recycle TCP/IP, then monitor CPU and paging rates to determine if they are within recommended system guidelines.

Repeat this process until you determine the ideal buffer size.

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

For more information about TCP/IP performance, see Chapter 5 of the Performance Capabilities Reference. Links to several editions of the Performance Capabilities Reference are in the Performance Management Resource Library.

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 the application server.

As part of configuring an application server, you can fine-tune settings that enhance system use of the JVM. For more information about JVM settings, see Java virtual machine settings.

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 *app_server_root*/java/bin directory. You can also check the `SystemOut.log` file for 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.

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.

You can adjust the following settings to change how the system uses the JVM. The following steps do not have to be performed in any specific order.

1. **Change the setting for class garbage collection**

   The class garbage collection argument, -Xnoclassgc, disables class garbage collection so that your applications can reuse classes more easily. You can monitor garbage collection using the -verbosegc configuration setting because its output includes class garbage collection statistics.

   To view or set this argument

   a. In the administrative console, click **Servers > Application Servers >** *server*.

   b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.

c. Enter `-Xnoclassgc` in the Generic JVM arguments field.
d. Click **Apply** or **OK**.
e. Save changes to the master configuration.
f. Stop and restart the application server.

Default:	Class garbage collection is enabled.
Recommended:	Do not disable class garbage collection.

2. **Change the value of the Initial heap size setting**

   The Initial heap size setting specifies, in megabytes, how often garbage collection runs, and can have a significant effect on performance. For more information, see the topic Tuning Garbage Collection for Java and WebSphere on iSeries in the i5/OS Information Center.

   a. In the administrative console, click **Servers > Application Servers >** *server*.
   b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.
   c. Specify a value in the Initial Heap Size field.
   d. Click **Apply** or **OK**.
   e. Save changes to the master configuration.
   f. Stop and restart the application server.

Default:	96 MB
Recommended:	96MB per processor

3. **Change the value of the Maximum heap size setting**

   The Maximum heap size setting specifies how often garbage collection runs. This setting can have a significant effect on performance. For more information, see the topic Tuning Garbage Collection for Java and WebSphere on iSeries in the i5/OS Information Center.

   a. In the administrative console, click **Servers > Application Servers >** *server*.
   b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.
   c. Specify a value in the Maximum Heap Size field.
   d. Click **Apply** or **OK**.
   e. Save changes to the master configuration.
   f. Stop and restart the application server.

Default:	0, which indicates that there is no maximum value.
Recommended:	It is recommended that you do not change the maximum heap size. When the maximum heap size triggers a garbage collection cycle, the i5/OS JVM's garbage collection stops operating asynchronously. When this happens, the application server cannot process user threads until the garbage collection cycle ends, which significantly lowers performance.

4. **Change the setting for the Just-In-Time (JIT) compiler**

   A Just-In-Time (JIT) compiler is a platform-specific compiler that generates machine instructions for each method as needed. For more information, see the sections Using the Just-In-Time compiler and Just-In-Time compiler in the i5/OS Information Center topic*IBM Developer Kit for Java*.

   a. In the administrative console, click **Servers > Application Servers >** *server*.
   b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.

c. Select the **Disable JIT** option if you want to disable the JIT.
d. Enter -Djava.compiler=jitc in the Generic JVM arguments field if you want to run with the full JIT compiler.
e. Click **Apply** or **OK**.
f. Save changes to the master configuration.
g. Stop and restart the application server.

Default:	JIT is enabled.
Recommended:	It is recommended that you do not disable the JIT compiler, and you enable the full JIT compiler (see step 4.d). The os400.jit.mmi.threshold can have a significant effect on performance. For more information about the JIT compiler and the os400.jit.mmi.threshold property, see the section Just-In-Time compiler in the i5/OS Information Center topic *IBM Developer Kit for Java*.

See Java memory tuning tips for additional tuning information.

If your application experiences slow response times at startup or first touch, you might want to use the Java user classloader cache. For more information, see Caching classes previously loaded by a user class loader.

Tuning transport channel services

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

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 transport channels associated with a transport chain can improve the performance of that chain.

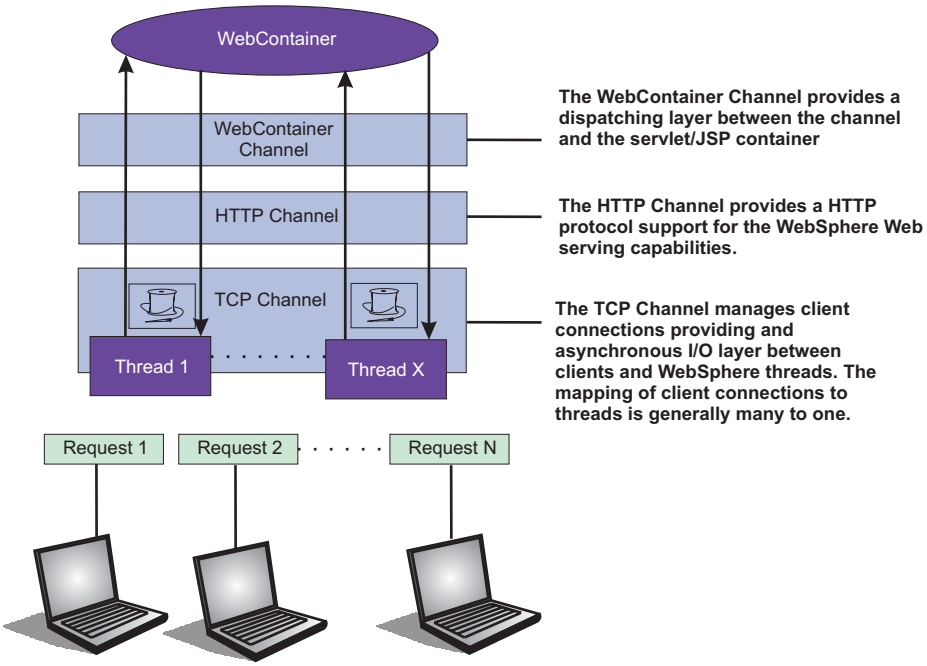


Figure 1. Transport Channel Service

- **Adjust TCP transport channel settings.** In the administration console, click **Servers > 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. Assign a thread pool to a specific HTTP port. Each TCP transport channel is assigned to a particular thread pool. Thread pools can be shared between one or more TCP transport channels as well as with other components. The default settings for a TCP transport channel is to have all HTTP based traffic assigned to the WebContainer thread pool and all other traffic assigned to the Default thread pool. Use the Thread pool pull-down to assign a particular thread pool to each TCP transport channel. The default settings for this parameter has all HTTP based traffic assigned to the WebContainer thread pool and all other traffic is assigned to the Default thread pool. (Thread pool collection describes how to create additional thread pools.)
  6. Tune the size of your thread pools. By default, a thread pool can have a minimum of 10 threads and a maximum of 50 maximum threads. To adjust these values, click on **Thread pools >** *threadpool_name* and adjust the values specified for the Minimum Size and Maximum Size parameters for that thread pool.

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

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

- **Adjust HTTP transport channel settings.** In the administration console, click **Servers > 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 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 need to be changed.
  2. Click on the Web container transport channel defined for that chain.
  3. If multiple writes are required to handle responses to the client, change the value specified for the Write buffer size parameter to a value that is more appropriate for your clients. The Write buffer size parameter controls the maximum amount of data per thread that the Web container buffers before sending the request on for processing. The default value is 32768 bytes, which is sufficient for most applications. If the size of a response is greater than the size of the write buffer, the response is chunked and written back in multiple TCP writes.

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

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

  To change the bounded buffer parameters:
  1. In the administrative console, click **Servers > Application Servers >** *server*.
  2. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.
  3. Specify one of the following parameters in the Generic JVM arguments field.
  4. Click **Apply** or **OK**.
  5. Enter one of the following custom properties in the Name field and an appropriate value in the Value field, and then click **Apply** to save the custom property and its setting.
     - com.ibm.ws.util.BoundedBuffer.spins_take=*value*

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

Default:	4
Recommended:	Any non-negative integer value is allowed. In practice an integer between 2 and 8 have shown the best performance results.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_take=6. Six attempts are made before the thread is suspended.

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

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

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

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

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

Default:	4
Recommended:	Any non-negative integer value is allowed. In practice an integer between 2 and 8 have shown the best performance results.
Usage:	com.ibm.ws.util.BoundedBuffer.spins_put=6. Six attempts are made before the thread is suspended.

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

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

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

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

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

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

- Click **Apply** and then **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.

For proper system sizing for WebSphere Application Server workloads, use the IBM Systems Workload Estimator.

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

- **Optimize disk speed**
  – **Description:** Disk speed and the number of disk arms have a significant effect on application server performance in the following cases:
    - Your application is heavily dependent on database support .
    - Your application uses messaging extensively.
  – **Recommendation:** Use disk I/O subsystems that are optimized for performance, for example, Redundant Array of Independent Disks (RAID). Distribute the disk processing across as many disks

as possible to avoid contention issues that occur with 1 or 2 disk systems. For more information about disk arms and how they can affect performance, see the iSeries Disk Arm Requirements documentation.

- **Increase processor speed and processor cache**
  - **Description:** In the absence of other bottlenecks, increasing the processing power can improve throughput, response times, or both. On WebSphere Application Server for i5/OS, processing power can be related to the Commercial Processing Workload (CPW) value of the system. For more information about CPW values, see the http://www-03.ibm.com/servers/eserver/iseries/perfmgmt/resource.html Web site.
- **Increase system memory**
  - **Description:** If a large number of page faults occur, performing the following tasks to improve performance:
    - Increase the memory available to WebSphere Application Server.
    - Move WebSphere Application Server to another memory pool.
    - Remove jobs from the WebSphere Application Server memory pool
  - **Recommendation:** To determine the current page fault level, run the Work with System Status (WRKSYSSTS) command from an i5/OS command line. For information about the minimum memory requirements, see the IBM Support Web site.
- **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.
- **Verify that the activity levels for storage pools are sufficient**
  - **Description:** Verify that the activity levels for storage pools are sufficient. Increasing these values can prevent threads from transitioning into the ineligible condition.
  - **Recommendation**
    - To modify the activity level for the storage pool in which you are running WebSphere Application Server, run the following **WRKSYSSTS** command from the command line:

      ```
      WRKSYSSTS ASTLVL(*INTERMED)
      ```
    - Perform the following steps to set the QMAXACTLVL system value to a value equal to or greater than the total activity level for all pools, or `*NOMAX`:
      - Run the following **WRKSYSSTS** command from the command line:

        ```
        WRKSYSSTS ASTLVL(*INTERMED)
        ```
      - Adjust the value in the **Max Active** column.

Tuning operating systems

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

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

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

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

Tuning i5/OS systems

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

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

For detailed performance tuning, refer to the Tune server performance topic in the iSeries Information Center.

This tuning procedure improves performance of WebSphere Application Server on the iSeries operating system. After tuning your operating system for performance, consult other tuning topics for various tuning tips.

Tuning Web servers

WebSphere Application Server provides plug-ins for several Web server brands and versions. If you are running your Web server on a non-i5/OS platform, see the product documentation for performance tuning information.

For additional information, refer to Chapter 6 of the Performance Capabilities Reference Manual. This manual is available in the Performance Management Resource Library.

The IBM HTTP Server (powered by Apache) is a multi-process, multi-threaded server. To tune this Web server:

- Enable the access logs. The access logs record all incoming HTTP requests. Logging can degrade performance even though logging occurs in a separate process from the Web server function.

  By default, the access log is disabled. It is recommended that you do not enable the access logs unless you need a record of all incoming HTTP requests.

  To enable the access logs:

  1. Open the IBM HTTP Server httpd.conf file, located in the /QIBM/ProdData/HTTPA/conf directory.
  2. Search for lines with the text CustomLog.
  3. Remove the hash mark (#) at the beginning of the line to enable a custom access log.
  4. Save and close the httpd.conf file.
  5. Stop and restart the IBM HTTP Server.
- Change the ThreadsPerChild directive setting. The ThreadsPerChild directive specifies the maximum number of concurrent client requests that the server processes at any time. The Web server uses one thread for each request that it processes. The value specified for this directive does not represent the number of active clients.

  To change the ThreadsPerChild directive setting:

  1. Open the IBM HTTP Server httpd.conf file, located in the /QIBM/ProdData/HTTPA/conf directory.
  2. Search for the ThreadsPerChild directive.
  3. Change the setting. The default value is 40. It is recommended that you either use the default value or increase the value if you need to increase the number of concurrent client requests that the server can process at any time. You should not decrease the setting of this directive.
  4. Save and close the httpd.conf file.
  5. Stop and restart the IBM HTTP Server.
- Change the ListenBackLog directive setting. This directive specifies the length of the pending connections queue. When several clients request connections to the IBM HTTP Server, and all threads are in use, a queue is created to hold additional client requests.

If you use the default Fast Response Cache Accelerator (FRCA) feature, the value specified for the ListenBackLog directive is ignored, because FRCA uses its own internal queue.

To change the ListenBackLog directive setting:

1. Open the IBM HTTP Server httpd.conf file, located in the /QIBM/ProdData/HTTPA/conf directory.
2. Search for the ListenBackLog directive.
3. Change the setting. For the IBM HTTP Server 1.3.26, the default setting is 1024 if FRCA is enabled, and 511 if FRCA disabled. It is recommended that you use these default values.
4. Save and close the httpd.conf file.
5. Stop and restart the IBM HTTP Server.

For more information about tuning heavily loaded Web servers, see Performance: Resources for learning

Tuning WebSphere applications

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

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

Product architecture and programming model, at a glance

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

Web applications

Session management tuning

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)

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

- The session manager invalidates sessions only at the scheduled time, therefore sessions are available to an application if they are requested before the session is invalidated.
- 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. See the *Administering applications and their environment* PDF for more information.
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.

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.

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.

EJB applications

EJB Container tuning

If you use applications that affect the size of the EJB Container Cache, it is possible that the performance of your applications can be impacted by an incorrect size setting. 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.

The following table lists which backend databases support batch update.

Table 1.

Database	Supports Batch update	Supports Batch update with Optimistic Concurrency Control
DB2	yes	no
Oracle	yes	no
DB2 Universal Driver	yes	yes
Informix	yes	yes
SQLServer	yes	yes
Cloudscape	yes	yes

Note: Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

Persistence Manager cache Tuning

Persistence Manager has two different types of caching mechanisms available: *legacy cache* and *two-level cache*. Normally two-level cache performs better than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property

```
com.ibm.ws.pm.useLegacyCache=false
```

Persistence Manager Partial Updates Tuning

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence Manager has two different types of caching mechanisms available, legacy cache and two-level cache. Normally, two-level cache performs better than legacy cache because of the optimizations in this mode. In certain applications where you need to perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both.

```
'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'
```

Web services

Tuning Web services security for Version 6.1 applications

The Java Cryptography Extension (JCE) is integrated into the software development kit (SDK) version 1.4.x and is no longer an optional package. However, the default Java Cryptography Extension (JCE) jurisdiction policy file shipped with the SDK enables you to use cryptography to enforce this default policy.

The Java Cryptography Extension (JCE) is integrated into the software development kit (SDK) version 1.4.x and is no longer an optional package. However, due to export and import regulations, the default Java Cryptography Extension (JCE) jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only. To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web services security might improve.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 5, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
2. Click **J2SE 5.0**
3. Scroll down and click **IBM SDK Policy files**.
   The Unrestricted JCE Policy files for the SDK Web site is displayed.
4. Click **Sign in** and provide your IBM intranet ID and password.
5. Select the appropriate Unrestricted JCE Policy files and then click **Continue**.
6. View the license agreement and then click **I Agree**.
7. Click **Download Now**.

For WebSphere Application Server platforms using the Sun-based Java Development Kit (JDK) Version 5, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://java.sun.com/j2se/1.5.0/download.jsp
2. Click **Other Downloads**.
3. Locate the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 1.5.1information and click **Download**. The policy files are downloaded onto your machine.

In IBM WebSphere Application Server Version 6.1, Web services security supports the use of cryptographic hardware devices. There are two ways in which to use hardware cryptographic devices with Web services security.

See Hardware cryptographic device support for Web Services Security for more information.

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

Tuning Web services security for Version 5.x applications

The Java Cryptography Extension (JCE) policy is integrated into the software development kit (SDK) Version 1.4.x and is no longer an optional package. However, due to export and import regulations, the default JCE jurisdiction policy file shipped with the SDK enables you to use strong, but limited, cryptography only. To enforce this default policy, WebSphere Application Server uses a JCE jurisdiction policy file that might introduce a performance impact. The default JCE jurisdiction policy might have a performance impact on the cryptographic functions that are supported by Web services security. If you have Web services applications that use transport level security for XML encryption or digital signatures, you might encounter performance degradation over previous releases of WebSphere Application Server. However, IBM and Sun Microsystems provide versions of these jurisdiction policy files that do not have

restrictions on cryptographic strengths. If you are permitted by your governmental import and export regulations, download one of these jurisdiction policy files. After downloading one of these files, the performance of JCE and Web Services security might improve.

For WebSphere Application Server platforms using IBM Developer Kit, Java Technology Edition Version 1.4.2, including the AIX, Linux, and Windows platforms, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://www.ibm.com/developerworks/java/jdk/security/index.html
2. Click **Java 1.4.2**
3. Click **IBM SDK Policy files**.

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

For WebSphere Application Server platforms using the Sun-based Java Development Kit (JDK) Version 1.4.2, including the Solaris environments and the HP-UX platform, you can obtain unlimited jurisdiction policy files by completing the following steps:

1. Go to the following Web site: http://java.sun.com/j2se/1.4.2/download.html
2. Click **Other Downloads**.
3. Locate the JCE Unlimited Strength Jurisdiction Policy Files 1.4.2 information and click **Download**. The policy files are downloaded onto your machine.

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

Setting tuning properties of a messaging engine

Use this task to set the tuning properties for a messaging engine.

You can set the following property to improve the performance of a messaging engine:

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

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

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

Messaging engine failover between v6 and v6.1

It is not permissible to failover a messaging engine using a file store onto a WebSphere Application Server v6 server. If you have a cluster as a bus member that consists of a mixture of v6 and v6.1 servers, you must modify the high availability policy to prevent this.

To prevent failover of a v6.1 messaging engine to a v6 server, the high availability policy for the messaging engine should be modified so that the cluster is effectively divided into sets of servers at the different versions and the messaging engine is restricted to the servers at v6.1.

Tuning and problem solving for messaging engine data stores

Obtain an overview of improving the performance of messaging engine data stores and understanding problems that can occur with a data store.

For more information about tuning and problem solving for messaging engine data stores, see the following topics:

- "Tuning the JDBC data source of a messaging engine"
- "Controlling the memory buffers used by a messaging engine"
- Sharing connections to benefit from one-phase commit optimization
- Diagnosing problems with data store exclusive access locks
- "Diagnosing problems with your data store configuration" on page 51
- "Avoiding failover problems when you use DB2 v8.2 with HADR as your data store" on page 52

Tuning the JDBC data source of a messaging engine

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

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

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

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

      Tip: If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.

   b. Set the **Purge policy** to *EntirePool*. This policy enables the connection pool to release all connections when the messaging engine stops.

Controlling the memory buffers used by a messaging engine

To control the sizes of the memory buffers used by a messaging engine you should follow these instructions and tips. Also learn about how to set the messaging engine to improve its interaction with its data store.

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

Name	Value
sib.msgstore.discardableDataBufferSize	The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is best effort nonpersistent. The messaging engine holds this data entirely within this memory buffer and never writes this data to the data store. When the messaging engine adds data to this buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. This behavior enables the messaging engine to discard best effort nonpersistent messages. The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises data both that is involved in active transactions, and any other best effort nonpersistent that the messaging engine has neither discarded nor consumed. The messaging engine can discard only data that is not involved in active transactions. **Tip:** If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transaction, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException. The **sib.msgstore.discardableDataBufferSize** property of the messaging engine controls the size of the discardable data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes.
sib.msgstore.cachedDataBufferSize	The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space. The **sib.msgstore.cachedDataBufferSize** property of the messaging engine controls the size of the cached data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes.
sib.msgstore.transactionSendLimit	The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100.

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

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

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

5. Save your changes to the master configuration.

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

Increasing the number of data store tables to relieve concurrency bottleneck

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

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

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

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

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

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

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

Diagnosing problems with your data store configuration

Find out how to diagnose problems that are caused by your data store configuration and possible solutions to these problems.

The following problems depend on the database that you use with your data store configuration and the level of that database:

- Examine this section if your messaging engine uses an Oracle 9i database for its data store and your messaging engine fails to start. If the messaging engine fails with the following message, where `XXXXXXXX` is the schema for the table, ensure that your Oracle installation is at 9.2.0.4, or higher:

  ```
  CWSIS1530E: The data type, 1,111, was found instead of the expected type, 2,004, for column,
  LONG_DATA, in table, XXXXXXXX.SIB000.
  ```

- Examine this section if your messaging engine uses a Sybase database for its data store. When you create your Sybase server:
  - Ensure that you create the database server with a page size of at least 4k.
  - Ensure that you set the **lock scheme** property on your server to the value *datarows*. This avoids the possibility of a deadlock on the data store tables.
- Examine this section if your messaging engine uses an Informix database for its data store and the messaging engine is unable to access its data store. When you configure your messaging engine to use an Informix database, ensure that you specify the schema name in lower case. For a full description of the configuring procedure, refer to Modify data store configurations.

Avoiding failover problems when you use DB2 v8.2 with HADR as your data store

Use this task to avoid problems that can occur when a messaging engine that is configured to use DB2 v8.2 with the High Availability Data Recovery (HADR) feature for its data store terminates if the DB2 database fails over.

If you use the High Availability Data Recovery (HADR) feature of DB2, note the following restrictions:

- The messaging engine default messaging provider supports only the synchronous and near-synchronoous synchronization modes of HADR. The default messaging provider does not support asynchronous HADR configurations.
- The TAKEOVER BY FORCE command is permitted only when the standby database is in peer state, or in a non-peer state (such as disconnected state) having changed from peer state.

One-phase commit optimization tuning

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

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

1. Open the administrative console.
2. Click on **Enterprise Applications** > *servername* > **Map data sources for all 2.x CMP beans**.
3. On the content pane, select the check boxes next to all the CMPs.
4. Select *Per application* in the **Resource authorization** selection list.
5. You can modify the messaging engine's resource authorization to *Per application* by modify the property file `sib.properties` by adding the custom property `sib.msgstore.jdbcResAuthForConnections=Application`.

Setting tuning properties for a mediation

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

Before you begin this task, you should review the guidance on when it is appropriate to tune a mediation for performance in the topic Guidance for tuning mediations for performance.

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

Name	Value
sib:SkipWellFormedCheck	Whether you want to omit the well formed check that is performed on messages after they have been processed by the mediation. Either true or false. **Note:** This property is overridden for messages that have the delivery option assured persistent, and a well formed check is always performed.

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

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

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

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

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

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

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

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

      Specify the following properties for the connection factory:

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

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

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

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

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

h. Click **OK**.

i. Save your changes to the master configuration.

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

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

   a. **Optional:** To determine the data source used by the messaging engine, click **Servers** → **Application servers** → **server_name** → **Messaging engines** → **engine_name** → **Data store** The **Data source name** field displays the name of the data source; by default:

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

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

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

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

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

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

      3) Click **Next**

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

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

6. Complete other panels as needed.

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

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

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

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

   d. Click **Finish**.

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

Tuning service integration technologies

Use this task to set tuning properties that control the performance of message-driven beans and other messaging applications deployed to use service integration technologies.

To optimize the performance of messaging with service integration technologies, such as message-driven beans that use the default messaging provider, you can use the following parameters set through the WebSphere administrative console or command line interfaces.

- Viewing the Available Message Count on a destination enables you to determine whether your message consumers are able to cope with your current workload. If the available message count on a given destination is too high, or is increasing over time, you should consider some of the tuning recommendations on this page.
  1. To monitor the available message count for a queue, you need to enable runtime AvailableMessageCount statistics for the queue. If you restart administrative server, you need to enable AvailableMessageCount statistics again because such runtime settings are not preserved when the server is restarted.

     To enable AvailableMessageCount statistics using the administrative console, complete the following steps:

     a. In the navigation pane, click **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)**
     b. In the content pane, click *server_name*
     c. Click the Runtime tab.
     d. In the Currently monitored statistic set, click **Custom**
     e. On the Custom monitoring level panel, click **SIB Service** → **SIB Messaging Engines** → *messageEngine_name* → **Destinations** → **Queues** → *queue_name*
     f. Select the AvailableMessageCount option.
     g. Click the **Enable** button at the top of the panel.
  2. To view the available message count, you can use the administrative console to complete the following steps:

     a. In the navigation pane, click **Monitoring and Tuning** → **Performance Viewer** → **Current activity**
     b. In the content pane, click *server_name*
     c. Click **Performance Modules** → **SIB Service** → **SIB Messaging Engines** → *messageEngine_name* → **Destinations** → **Queues** → *queue_name*
     d. Click the **View Module(s)** button at the top of the Resource Selection panel, located on the left side. This displays the AvailableMessageCount data in the Data Monitoring panel, located on the right side.

        You can use the Data Monitoring panel to manage the collection of monitoring data; for example, you can use the buttons to start or stop logging, or to change the data displayed as either a table or graph.
- Monitoring MDB Thread Pool Size for the Default Message Provider. You may experience a performance bottleneck if there are insufficient threads available for the Message Driven Beans. There is a trade-off between providing sufficient threads to maximize the throughput of messages and configuring excessive threads, which can lead to CPU starvation of the threads in the application server. If you notice that the throughput for express nonpersistent, reliable nonpersistent, or reliable persistent messaging has fallen as a result of increasing the size of the default thread pool, then you should decrease the size of the thread pool and reassess the message throughput.
  1. By default MDBs use the default thread pool. To view or change the number of threads in the default thread pool for an application server, you can use the administrative console to complete the following steps:

     a. In the navigation pane, click **Servers** → **Application servers**
     b. In the content pane, click *server_name*

c. Under Additional properties, click **Thread Pools** → **Default**. By default the Minimum size value is set to 5 and the Maximum size value is set to 20. The best performance is obtained by setting the Maximum size value to the expected maximum concurrency for all message-driven beans. For high throughput using a single message-driven bean, 41 was found to be the optimal Maximum size value.

d. To change the Maximum size value, type the new value in the Maximum size field then click **OK**. Finally, save your changes to the master configuration.

2. As the default thread pool is also used by other WAS components it can be beneficial to define a separate thread pool for the MDBs. This will reduce thread contention for the default thread pool. To create your own thread pool you can use the administrative console to complete the following steps:

   a. In the navigation pane, click **Servers** → **Application servers**

   b. In the content pane, click *server_name*

   c. Under Additional properties, click **Thread Pools**. Create a new thread pool. Create sufficient threads to support the maximum amount of concurrent work for the MDBs.

   d. b. Change the SIB JMS Resource Adapter to use the new thread pool: **Resources** -> **Resource Adapters** -> **Resource Adapters**.

   e. Open **Preferences** and select the **SIB JMS Resource Adapter** with the appropriate scope depending upon the scope of the connection factories. Add the name of the new thread pool in the **Thread pool alias** box. Click **Apply** and save the changes.

- Tuning MDB performance with the default messaging provider.

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

     a. Click on **Resources** -> **JMS** -> **Activation Specification**.

  2. 2. Delivering batches of messages to each MDB endpoint can improve performance particularly when used with Acknowledge mode set to Duplicates-ok auto-acknowledge. This parameter is applicable to MDBs using an activation specification. If message-ordering must be retained across failed deliveries, the batch size should be set to 1. This parameter can be set from the administrative console:

     a. Click on **Resources** -> **JMS** -> **Activation Specification**.

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

- Reducing the number of OutOfCacheSpace errors in the SystemOut.log file.

  OutOfCacheSpace errors in the SystemOut.log file indicate that the discardable data buffer used by the messaging engine is overflowing. For best effort nonpersistent messages, the messaging engine starts to discard messages when this buffer is full. You can increase the size of this data buffer to allow more best effort nonpersistent data to be handled before the messaging engine begins to discard the messages.

  For more information about tuning the size of the discardable data buffer, set by the sib.msgstore.discardableDataBufferSize property of a messaging engine, see “Controlling the memory buffers used by a messaging engine” on page 49.

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

- Increase the heap size for the Java Virtual Machine (JVM) used by the WebSphere Application Server by setting the Initial Heap Size and Maximum Heap Size properties of the application server. To view the administrative console page, click **Servers** → **Application Servers** → ***server_name*** → **Server Infrastructure** → **Process Definition** → **Java Virtual Machine**. For more information about changing the JVM configuration for the application server, see Java virtual machine settings.
- Reduce the cumulative size of the set of messages being processed within the transaction.

- Changing the maximum connections in a Connection Factory for the default messaging provider. The maximum connections parameter limits the number of local connections. The default is 10. This parameter should be set to a number equal to or greater than the number of threads (enterprise beans) concurrently sending messages. Using the administrative console you can set the Maximum connections property as follows:
  1. Click on **Resources** -> **JMS** -> **Topic Connection Factory** -> *factory_name* -> **Connection pool properties**
  2. Enter the required value in the **Maximum connections** field.
  3. Click **Apply** and save the changes to master configurations.
- Tuning the messaging engine message stores
  - For file store configurations see File stores .
  - For tuning information of JDBC data sources see “Tuning and problem solving for messaging engine data stores” on page 49
- Additional tuning advice for a messaging engine using a JDBC data source.

  To improve the performance of messaging throughput of a messaging engine data store, you can tune the JDBC connection pool and statement cache size. In tests of high throughput MDB workloads, the following changes provided a 10% gain in throughput.
  1. The messaging engine uses a connection pool for managing the JDBC connections to its data store. Tuning the size of the pool can improve the messaging throughput.

     To view or change the size of the connection pool, you can use the administrative console to complete the following steps:
     a. In the navigation pane, click **Resources** → **JDBC Providers**
     b. In the content pane, click *jdbc_provider_name*
     c. Under Additional properties, click **Data sources** → *data_source_name*
     d. Under Additional properties, click **Connection pool properties**
     e. View the Maximum connections property and the Minimum connections property. By default, these properties are set to Maximum connections=10 and Minimum connections=1. Setting the value of both these properties to 50 is recommended. For especially high throughput workloads, setting the value of both these properties up to 100 can be beneficial. You may need to configure the underlying database to accept this many concurrent connections.
     f. To change the value of a property, type a new value in the property field then click **OK**. Finally, save your changes to the master configuration.
  2. The statement cache contains recently used prepared statements to remove the costs associated with repeated preparation of statements. Tuning the size of the cache helps prevent useful entries from being discarded to make room for new entries.

     To view or change the size of the statement cache, you can use the administrative console to complete the following steps:
     a. In the navigation pane, click **Resources** → **JDBC Providers**
     b. In the content pane, click *jdbc_provider_name*
     c. Under Additional properties, click **Data sources** → *data_source_name*
     d. Under Additional properties, click **WebSphere Application Server data source properties**
     e. View the Statement cache size property. By default, the value of this property is set to 10. For high throughput JMS messaging, a value of 40 is recommended.

f. To change the value of the property, type a new value in the property field then click **OK**. Finally, save your changes to the master configuration.

- Tuning reliability levels for messages.

  The reliability level chosen for the messages has a significant impact on performance. In order of decreasing performance (fastest first), the reliability levels are: Best-Effort Nonpersistent, Express Nonpersistent, Reliable Nonpersistent, Reliable Persistent, and Assured Persistent. For MDB point-to-point messaging, best-effort nonpersistent throughput is more than 6 times greater than assured persistent.

  For more information about reliability levels, see Message reliability levels.

- Tuning MDB performance with the default messaging provider.

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

Tuning the SIBWS

You can use the administrative console or a Jacl script to tune performance settings for the service integration bus Web services enablement (SIBWS).

The SIBWS dynamically selects an optimized route through the code where possible. This fast-path route through the bus is used if the following criteria are met:

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

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

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

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

To improve the performance of the SIBWS, you can tune the following parameters:

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

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

To tune the SIBWS, complete one of the following two steps:

- Use the administrative console to tune the SIBWS, or
- Use a Jacl script to tune the SIBWS.

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

- Insert the header schemas into the SDO repository.
- **Optional:** To use the administrative console to tune the SIBWS, complete the following steps:
  1. Use the topic Tuning Java virtual machines to set the JVM heap size to a larger value than the default value (256 megabytes). The value should generally be as large as possible without incurring paging.
  2. Use the topic Tuning service integration messaging to tune the maximum number of instances of a message-driven bean, the maximum batch size for batches of messages for a bean, and the number of threads available to service requests for a bean.
  3. Use the topic Tuning the application serving environment to tune the general application serving environment, in particular the size of the Web Container Thread Pool. In a server which is exclusively serving requests to the SIBWS the default thread pool and the Web Container thread pool should be the same size.
  4. Use the topic Configuring the mediation thread pool to configure the number of threads available to concurrent mediations.
- To use a Jacl script to tune the SIBWS, use the wsadmin scripting client (from within Qshell) to run a script based on the following example:

```
#-------------------------------------------------------------------
# SIBWS WebSphere Tuning Script
#-------------------------------------------------------------------
##
# This script is designed to modify some of the tuning pertinent to a SIBWS
# deployment.
# In order to tune the config parameters, simply change the values
# provided below. This script assumes that all server names in a
# cluster configuration are unique.
#
# To invoke the script, type:
#   wsadmin -f tuneWAS.jacl <scope> <id>
#      scope      - 'cluster' or 'server'
#      id         - name of target object within scope (i.e. servername)
#
# Example:
#   wsadmin -f tuneWAS.jacl server server1
#
#
#-------------------------------------------------------------------
$AdminConfig setValidationLevel NONE

puts "Starting script..."
puts "Reading config parameters..."

#-------------------------------------------------------------------
# COMMON CONFIG PARAMETERS
# - Adjust these parameters based on the intended target system (Defaults in parentheses)
#-------------------------------------------------------------------
# WebContainer Thread Pool (10,50)
set minWebPool     10
set maxWebPool     15

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

```
# Mediations Thread Pool (1,5)
set minMediationPool 10
set maxMediationPool 15

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

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

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

# J2CActivationSpec for the SIB_RA Resource adapter
set SIB_RA_maxConcurrency 15
set SIB_RA_maxBatchSize   5

# Java2 Security (false for 5.1 and true for 6.0)
set j2Security     false

# Parallel server startup
set parallelStart  false


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

proc printUsageAndExit {} {
   puts " "
   puts "Usage: wsadmin -f tuneWAS.jacl <cluster | server> <name>"
   exit
}

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

proc getName {objectid} {
   set endIndex [expr [string first "(" $objectid] - 1]

   return [string range $objectid 0 $endIndex]
}


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

puts "Parsing command line arguments..."

if {[llength $argv] < 2} {
   printUsageAndExit
} else {
   set scope [lindex $argv 0]
   puts "Scope:   ${scope}"

   if {$scope == "cluster"} {
      set clustername [lindex $argv 1]
      puts "Cluster: ${clustername}"
   } elseif {$scope == "server"} {
      set servername [lindex $argv 1]
```

```
      puts "Server:  ${servername}"
   } else {
      puts "Error: Invalid Argument ($scope)"
      printUsageAndExit
   }
}

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

puts ""
puts "Obtaining server list..."

if {$scope == "cluster"} {
   set cluster [$AdminConfig getid "/ServerCluster:${clustername}/"]
   set temp [$AdminConfig showAttribute $cluster members]
   set memberList [split [string trim $temp "{ }"] " "]
   foreach member $memberList {
      set memberName [getName $member]
      lappend serverList [$AdminConfig getid "/Server:${memberName}/"]
   }
} else {
   set server [$AdminConfig getid "/Server:${servername}/"]
   lappend serverList $server
}


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

puts ""
puts "WebSphere configuration"
puts "-----------------------"
puts ""
puts "   Enforce Java2 Security:      ${j2Security} "
puts ""

puts "Servers:"
foreach server $serverList {
   puts "   [getName $server]"
}
puts ""
puts " Web ---------------------------------------------"
puts "   Min WebContainer Pool Size:  ${minWebPool} "
puts "   Max WebContainer Pool Size:  ${maxWebPool} "
puts " JVM ---------------------------------------------"
puts "   Min JVM Heap Size:           ${minHeap} "
puts "   Max JVM Heap Size:           ${maxHeap} "
puts "   Verbose GC:                  ${verboseGC}"
puts ""


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

# Accessing cell based security config
puts "Accessing security configuration..."
set sec [$AdminConfig list Security]
set attrs [subst {{enforceJava2Security $j2Security}}]
puts "Updating security..."
$AdminConfig modify $sec $attrs
```

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

foreach server $serverList {
   set servername [getName $server]
   puts ""
   puts "Server: $servername"
   puts ""


   # Accessing server startup config
   puts "Accessing server startup configuration..."
   puts "Parallel Startup (old/new):  [$AdminConfig showAttribute $server parallelStartEnabled]/
$parallelStart"
   set attrs [subst {{parallelStartEnabled $parallelStart}}]
   puts "Updating server startup..."
   puts ""
   $AdminConfig modify $server $attrs

   # Accessing web container thread pool config
   puts "Accessing web container thread pool configuration..."
   set tpList [$AdminConfig list ThreadPool $server]

   set oI [lsearch -glob $tpList "*WebContainer*"]
   set webPool [lindex $tpList $oI]
   puts "ThreadPool MaxSize (old/new):             [$AdminConfig showAttribute $webPool maximumSize
]/$maxWebPool"
   puts "ThreadPool MinSize (old/new):             [$AdminConfig showAttribute $webPool minimumSize
]/$minWebPool"
   puts "ThreadPool Inactivity Timeout (old/new):  [$AdminConfig showAttribute $webPool
inactivityTimeout]/$inactivity"
   set attrs [subst {{maximumSize $maxWebPool} {minimumSize $minWebPool} {inactivityTimeout
$inactivity}}]
   puts "Updating web container thread pool..."
   puts " "
   $AdminConfig modify $webPool $attrs

   # Accessing default thread pool config
   puts "Accessing default thread pool configuration..."
   set tpList [$AdminConfig list ThreadPool $server]

   set oI [lsearch -glob $tpList "*Default*"]
   set webPool [lindex $tpList $oI]
   puts "ThreadPool MaxSize (old/new):             [$AdminConfig showAttribute $webPool maximumSize
]/$maxDefaultPool"
   puts "ThreadPool MinSize (old/new):             [$AdminConfig showAttribute $webPool minimumSize
]/$minDefaultPool"
   puts "ThreadPool Inactivity Timeout (old/new):  [$AdminConfig showAttribute $webPool
inactivityTimeout]/$inactivity"
   set attrs [subst {{maximumSize $maxDefaultPool} {minimumSize $minDefaultPool} {inactivityTimeout
$inactivity}}]
   puts "Updating default thread pool..."
   puts " "
   $AdminConfig modify $webPool $attrs

   # Creating Mediations Thread Pool
   puts "Creating Mediations thread pool"
   set me [$AdminConfig list SIBMessagingEngine]
   set mtpName [$AdminConfig showAttribute $me name]-mediationThreadPool
   set tpAttrs [subst {{name $mtpName} {minimumSize $minMediationPool} {maximumSize
$maxMediationPool}}]
   puts "ThreadPool Name    : $mtpName"
   puts "ThreadPool MaxSize : $maxMediationPool"
   puts "ThreadPool MinSize : $minMediationPool"
   $AdminConfig create ThreadPool $me $tpAttrs mediationThreadPool
   puts "Mediations Thread Pool Created"
```

```
   puts " "

   # Accessing HTTP keepalive config
   puts "Accessing HTTP KeepAlive configuration..."
   set HTTPInbound [$AdminConfig list HTTPInboundChannel $server]

   set oI [lsearch -glob $HTTPInbound "*HTTP_2*"]
   set http2 [lindex $HTTPInbound $oI]
   puts "KeepAlive Enabled (old/new):        [$AdminConfig showAttribute $http2 keepAlive]/
$keepAliveEnabled"
   puts "Max Persistent Requests (old/new):  [$AdminConfig showAttribute $http2
maximumPersistentRequests]/$maxPersistentRequests"
   set attrs [subst {{keepAlive $keepAliveEnabled} {maximumPersistentRequests
$maxPersistentRequests}}]
   puts "Updating HTTP KeepAlives..."
   puts " "
   $AdminConfig modify $http2 $attrs

   # Accessing JVM config
   puts "Accessing JVM configuration..."
   set jvm [$AdminConfig list JavaVirtualMachine $server]
   puts "Initial Heap Size (old/new):  [$AdminConfig showAttribute $jvm initialHeapSize]/$minHeap"
   puts "Maximum Heap Size (old/new):  [$AdminConfig showAttribute $jvm maximumHeapSize]/$maxHeap"
   puts "VerboseGC Enabled (old/new):  [$AdminConfig showAttribute $jvm
verboseModeGarbageCollection]/$verboseGC"
   set attrs [subst {{initialHeapSize $minHeap} {maximumHeapSize $maxHeap}
{verboseModeGarbageCollection $verboseGC} }]
   puts "Updating JVM..."
   puts " "
   $AdminConfig modify $jvm $attrs

   # Accessing J2CActivationSpec for the SIB Resource Adapter
   puts "Modifying the J2CActivationSpec for the SIB Resource Adapter"
   set actSpec [$AdminConfig getid /J2CActivationSpec:SIBWS_OUTBOUND_MDB/]
   set propSet [$AdminConfig showAttribute $actSpec resourceProperties]

   set propSet [lindex $propSet 0]

   set maxConcurrency [list value $SIB_RA_maxConcurrency]
   set maxConcurrency [list $maxConcurrency ]

   set maxBatchSize   [list value $SIB_RA_maxBatchSize]
   set maxBatchSize   [list $maxBatchSize]

   foreach propId $propSet {
      if { [string compare [$AdminConfig showAttribute $propId name] maxConcurrency] == 0} {
    $AdminConfig modify $propId $maxConcurrency
    puts "Custom property changed : [$AdminConfig showall $propId] "
      }
      if { [string compare [$AdminConfig showAttribute $propId name] maxBatchSize] == 0} {
    $AdminConfig modify $propId $maxBatchSize
    puts "Custom property changed : [$AdminConfig showall $propId] "
      }
   }
   puts "J2CActivationSpec modifications complete"


}


puts ""
puts "Script completed..."
puts "Saving config..."
$AdminConfig save
```

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

Including SOAP header schemas in the SDO repository

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

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

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

Here is an example of an associated header schema:

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

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

1. Create a Jacl script called `sdoXSDImport.jacl` that contains the following code:

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

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

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

2. Use the wsadmin scripting client (from within Qshell) to insert the schema into the SDO repository by entering the following command:

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

   where

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

Setting tuning properties for service integration

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

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

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

To set tuning properties using the administrative console, click the relevant link from the following list:

- “Setting tuning properties of a messaging engine” on page 48.
- “Controlling the memory buffers used by a messaging engine” on page 49.

You can also use the administrative console to tune the data source. Refer to “Tuning the JDBC data source of a messaging engine” on page 49.

To set tuning properties for any of the components mentioned above by editing the sib.properties file, refer to Setting tuning properties by editing the sib.properties file.

Sub-topics

Setting tuning properties of a messaging engine

Use this task to set the tuning properties for a messaging engine.

You can set the following property to improve the performance of a messaging engine:

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

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

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

Controlling the memory buffers used by a messaging engine

To control the sizes of the memory buffers used by a messaging engine you should follow these instructions and tips. Also learn about how to set the messaging engine to improve its interaction with its data store.

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

Name	Value
sib.msgstore.discardableDataBufferSize	The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is best effort nonpersistent. The messaging engine holds this data entirely within this memory buffer and never writes this data to the data store. When the messaging engine adds data to this buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. This behavior enables the messaging engine to discard best effort nonpersistent messages. The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises data both that is involved in active transactions, and any other best effort nonpersistent that the messaging engine has neither discarded nor consumed. The messaging engine can discard only data that is not involved in active transactions. **Tip:** If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transaction, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException. The **sib.msgstore.discardableDataBufferSize** property of the messaging engine controls the size of the discardable data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes.
sib.msgstore.cachedDataBufferSize	The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space. The **sib.msgstore.cachedDataBufferSize** property of the messaging engine controls the size of the cached data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes.
sib.msgstore.transactionSendLimit	The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100.

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

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

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

5. Save your changes to the master configuration.

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

Tuning the JDBC data source of a messaging engine

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

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

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

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

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

Data access resources

EJB Container tuning

If you use applications that affect the size of the EJB Container Cache, it is possible that the performance of your applications can be impacted by an incorrect size setting. 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.

The following table lists which backend databases support batch update.

Table 2.

Database	Supports Batch update	Supports Batch update with Optimistic Concurrency Control
DB2	yes	no
Oracle	yes	no
DB2 Universal Driver	yes	yes
Informix	yes	yes
SQLServer	yes	yes
Cloudscape	yes	yes

Note: Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

Persistence Manager cache Tuning

Persistence Manager has two different types of caching mechanisms available: *legacy cache* and *two-level cache*. Normally two-level cache performs better than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property

```
com.ibm.ws.pm.useLegacyCache=false
```

Persistence Manager Partial Updates Tuning

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence Manager has two different types of caching mechanisms available, legacy cache and two-level cache. Normally, two-level cache performs better than legacy cache because of the

optimizations in this mode. In certain applications where you need to perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both.

`'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'`

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. If you run DB2 UDB for iSeries, consult the "DB2 Universal Database performance tips" article as a starting point reference.

DB2 Universal Database performance tips

You can easily adjust your system QSQSRVR prestart job settings to optimize the process of acquiring connections from DB2 Universal Database for i5/OS. On the i5/OS platform, QSQSRVR jobs process Java Database Connectivity (JDBC) tasks. By default, five QSQSRVR jobs are initially active. When fewer than two QSQSRVR jobs are unused, the i5/OS system creates two more jobs. If you increase the active job values, an application that establishes a large number of database connections over a short period of time might create connections more quickly. Increase the values for the initial number of jobs, threshold, and additional number of jobs by running this command on an i5/OS command line:

```
CHGPJE SBSD(QSYS/QSYSWRK) PGM(QSYS/QSQSRVR)
```

Do not start more QSQSRVR jobs than your application requires. Active QSQSRVR jobs require some overhead, even if these jobs are not used.

Related Web resources

The following links direct you to IBM Web pages for DB2 and i5/OS tuning tips that potentially can increase the performance of WebSphere Application Server applications.

- http://publib.boulder.ibm.com/iseries/v5r2/ic2924/index.htm?info/rzahf/rzahfmonitortunedb.htm This URL takes you to the ″Monitor and tune database performance″ section in the Database topic of the IBM iSeries Information Center. The content addresses DB2 Universal Database for i5/OS. If you use a different database, refer to that vendor documentation.
- http://www.ibm.com/servers/eserver/iseries/perfmgmt/resource.htm This URL takes you to the i5/OS Performance Management Resource Library page, which contains links to several editions of the Performance Capabilities Reference. Consult chapter 4 of this reference document.

Security

Security cache properties

The following Java virtual machine (JVM) 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.

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.

You can specify these system properties by completing the following steps:

1. Click **Servers > Application servers >** *server_name*.
2. Click **Java and Process Management > Process Definition**.
3. Under Additional properties, click **Java Virtual Machine**.

4. Specify the property name and its value in the Generic JVM arguments field. You can specify multiple property name and value pairs delimited by a space.

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 can 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 create 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, hardening, and maintaining

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

The following topics are covered in this section:

- Tuning security configurations
- Hardening security configurations
- Changing keys and passwords
- Securing passwords in files

Tuning security configurations:

Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you can disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your Java 2 Platform, Enterprise Edition (J2EE) roles? These questions are things to consider when designing your security infrastructure.

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

    See the article “Security cache properties” on page 71 for a list of these properties.

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

• Consider the following steps to tune Common Secure Interoperability version 2 (CSIv2).
- Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.
- If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
- Consider putting only an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use identity assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk (*) is used, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.
- Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.
- V6.0.x If you are communicating only with WebSphere Application Server Version 5 or higher servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

  Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

• Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.
1. In the administration console, click **Security > Secure administration, applications, and infrastructure**.
2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry** and click **Configure**.
3. Select the **Ignore case for authorization** option in the standalone LDAP registry configuration, when case-sensitivity is not important.
4. Select the **Reuse connection** option.
5. Use the cache features that your LDAP server supports.
6. Choose either the IBM Tivoli Directory Server or SecureWay directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.

7. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory can increase performance in group membership lookup. However, use **Role** only for group mechanisms.

- Consider the following steps to tune Web authentication.
  - Increase the cache and token timeout values if you feel your environment is secure enough. The Web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials that are already created. A disadvantage of increasing the token timeout is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.

    See the article “Security cache properties” on page 71 for a list of these properties.
  - Enable single sign-on (SSO). To configure SSO, click **Security > Secure administration, applications, and infrastructure**. Under Web security, click **Single sign-on (SSO)**.

    SSO is only available when you configure **LTPA** as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 6.1 and does not support SSO. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. Some situations exist where SSO is not a desirable and you do not want to use it in those situations.
  - Disable or enabling the **Web Inbound Security Attribute Propagation** option on the Single sign-on (SSO) panel if the function is not required. In some cases, having the function enabled can improve performance. This improvement is most likely for higher volume cases where a considerable number of user registry calls reduces performance. In other cases, having the feature disabled can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.
- Consider the following steps to tune authorization.
  - Map your users to groups in the user registry. Associate the groups with your Java 2 Platform, Enterprise Edition (J2EE) roles. This association greatly improves performance when the number of users increases.
  - Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
  - Judiciously assign security-constraints for servlets. For example, you can use the `*.jsp` URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the `*.jsp`, `*.do`, `*.html` extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.

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

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

Secure Sockets Layer performance tips:

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

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

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

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

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

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

  Note: HTTP Transports have been deprecated. For instructions on how to set a maximum keep alive value for channel based configurations, see HTTP transport channel settings.
- Increase the maximum number of requests per keep alive connection. The default value is 100, which means the application server closes the connection from the plug-in after 100 requests. The plug-in then has to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and preventing continuous send requests to tie up threads in the application server.
- Use a hardware accelerator if the system performs several SSL handshakes.

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

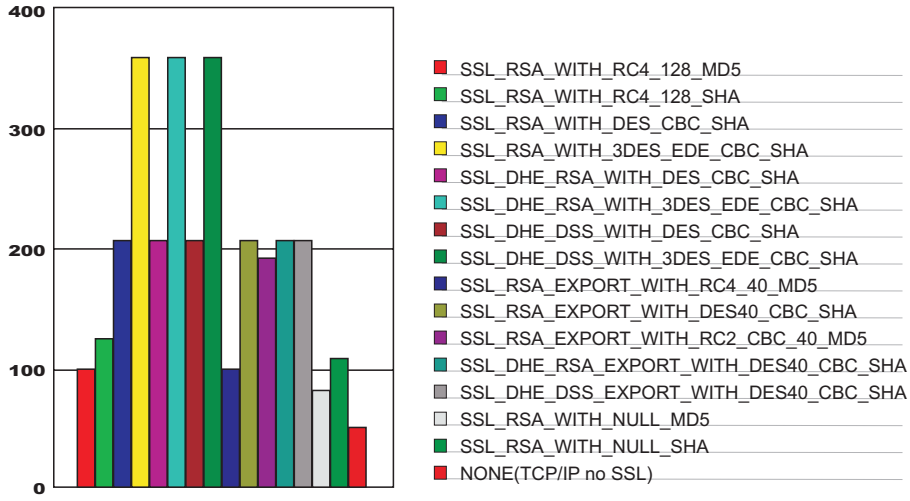
  The IBM Cryptographic Coprocessor is not supported for use with WebSphere Application Server. However, you can use the IBM Cryptographic Coprocessor to improve SSL performance for other products, such as IBM HTTP Server for iSeries, which is powered by Apache.
- Use an alternative cipher suite with better performance.

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

  The performance of bulk encryption and decryption is affected by the cipher suite used for an individual SSL connection. The following chart displays the performance of each cipher suite. The test software

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

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



An analysis of the above data reveals the following:

- Bulk encryption performance is only affected by what follows the WITH in the cipher suite name. This is expected since the portion before the WITH identifies the algorithm used only during the SSL handshake.
- MD5 and Secure Hash Algorithm (SHA) are the two hash algorithms used to provide data integrity. MD5 is generally faster than SHA, however, SHA is more secure than MD5.
- DES and RC2 are slower than RC4. Triple DES is the most secure, but the performance cost is high when using only software.
- The cipher suite providing the best performance while still providing privacy is SSL_RSA_WITH_RC4_128_MD5. Even though SSL_RSA_EXPORT_WITH_RC4_40_MD5 is cryptographically weaker than RSA_WITH_RC4_128_MD5, the performance for bulk encryption is the same. Therefore, as long as the SSL connection is a long-running connection, the difference in the performance of high and medium security levels is negligible. It is recommended that a security level of high be used, instead of medium, for all components participating in communication only among WebSphere Application Server products. Make sure that the connections are long running connections.

Tuning security:

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

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

- Disable security on any application servers that do not need security. You can disable security in the administrative console by clicking **Security > Secure administration, applications, and infrastructure** and deselecting the **Enable administrative security** option.

- Fine-tune the **Authentication cache timeout** value on the Authentication mechanisms and expiration panel in the administrative console. For more information, see the Secure administration, applications, and infrastructure settings topic.
- Configure the security cache properties. For more information, see the “Security cache properties” on page 71 topic.
- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console. For more information, see the Session management settings topic.
- Improve the performance of Web services security by downloading a Java Cryptography Extension (JCE) unlimited jurisdiction policy file that does not have restrictions on cryptography strength. For more information, see the “Tuning Web services security for Version 6.1 applications” on page 46 topic.
- Read the Secure Sockets Layer performance tips and “Tuning security configurations” on page 72 topics for more information.

Hardening security configurations:

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

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

- Take preventative measures to protect the infrastructure.
- Make applications less vulnerable to attack.

Securing passwords in files:

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

The following topics are covered in this section:

- Password encoding and encryption
- Encoding passwords in files
- Enabling custom password encryption

Password encoding and encryption:

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

By default, passwords are automatically encoded with a simple masking algorithm in various WebSphere Application Server ASCII configuration files. Additionally, you can manually encode passwords in properties files that are used by Java clients and by administrative commands for WebSphere Application Server.

The default encoding algorithm is referred to as XOR. An alternate OS400 encoding algorithm can be used with WebSphere Application Server for i5/OS that exploits native validation list (*VLDL) objects only. With the OS400 algorithm, passwords are stored in an encrypted form within a validation list. The configuration files contain indexes to the stored passwords instead of the masked passwords, as is done with the XOR algorithm.

Encoded passwords use the following syntax:

```
{algorithm}encoded_password
```

where {algorithm} is a tag that specifies the algorithm that is used to encode the password, which is either XOR or OS400. The *encoded_password* variable is the encoded value of the password. When a server or client needs to decode a password, it uses the tag to determine what algorithm to use and then uses that algorithm to decode the encoded password.

Java clients use passwords from the sas.client.props file, which is in the *profile_root*/properties directory.

To use password encoding with Java clients, the passwords must be manually encoded in the sas.client.props file using the **PropFilePasswordEncoder** tool.

The administrative commands for WebSphere Application Server use passwords from the soap.client.props file, which is also located in the /properties subdirectory, for SOAP connections. Some administrative commands optionally use passwords from the sas.client.props file in the /properties subdirectory for Remote Method Invocation (RMI) connections. To use password encoding with administrative commands, you must manually encode the passwords in the soap.client.props and sas.client.props files using the **PropFilePasswordEncoder** tool.

Attention: Whether you select to use the OS400 encoding algorithm or the default encoding algorithm, encoding is not sufficient to fully protect passwords. Native security is the primary mechanism for protecting passwords that are used in the configuration and property files for WebSphere Application Server.

Issues to consider when you use the OS400 password encoding algorithm

The following issues are important for you to consider before deciding to use the OS400 password encoding algorithm:

- You must set the QRETSVRSEC operating system value to 1 to use on the system that hosts the Java client application or WebSphere Application Server. With this setting, WebSphere Application Server to retrieve the encrypted passwords from the validation list.

  Attention: The QRETSVRSEC system value affects access to the encrypted data in all of the validation lists on your operating system. Do not use the OS400 password encoding algorithm if this setting is not consistent with your security policy for your operating system.
- You can use the OS400 algorithm with server instances only when all of the server instances within the administrative domain for WebSphere Application Server reside on the same i5/OS system. Consider the following related issues:
  - Administrative domains for WebSphere Application Server can extend across multiple i5/OS systems. You can use the OS400 password algorithm only when all of the servers within an administrative domain reside on the same i5/OS system.
  - Server configuration XML files contain encoded passwords. If the passwords that are contained in the XML files are encoded using the OS400 encoding algorithm, those encodings are valid only for the Application Server instances on the same i5/OS system on which the passwords were originally encoded. Copies of configuration files that contain passwords that are encoded using the OS400 encoding algorithm cannot be used to configure servers on other i5/OS systems.
  - All server instances within an administrative domain must be configured to use the same native validation list (*VLDL) object.
- For Java clients, you can use the OS400 password algorithm on any i5/OS system. However, option 1 must be installed on the system that hosts the Java client.
- If an error occurs while a password is encoded using the OS400 encoding algorithm, the XOR encoding algorithm is used to encode the password. An error might occur if an administrator manually creates the validation list object and grants insufficient authority to the validation list object for the i5/OS QEJB user profile.

Object and file security:

This topic discusses the various objects and files that contain sensitive information and need to be protected.

Secure integrated file system files

In addition to enterprise beans and servlets, WebSphere Application Server accesses integrated file system stream files. The following files might contain sensitive information. It is recommended that you give these files close consideration to ensure that unauthorized access is not granted.

- In the /properties subdirectory of your profile, the following files can contain user IDs and passwords:
  - sas.client.props
  - soap.client.props
  - sas.stdclient.properties
  - sas.tools.properties
  - wsserver.key

  By default, the /properties subdirectory is located in the *profile_root* directory. Each of the previous files is shipped with *PUBLIC authority set to *EXCLUDE. The QEJBSVR user profile is granted *RW authority to these files. Additional protection is available through password encoding. For more information, see “Password encoding and encryption” on page 77.
- In the /etc subdirectory if your profile, protect all of the key (KDB) files and trust (JKS) files that you create for your WebSphere Application Server profile.

  For the JKS files, the QEJBSVR user profiles should have *R authority and *PUBLIC should have *EXCLUDE authority.

  For the KDB files, the user profile that the Web server is running under should have *RX authority and *PUBLIC should have *EXCLUDE authority.

Secure database resources for WebSphere Application Server

WebSphere Application Server uses tables to persist data for user applications such as enterprise beans persistence and servlet session data. You have several options for controlling which user profiles are allowed access to this user data. For more information, see Database access security.

Secure WebSphere Application Server files

When you enable WebSphere Application Server security, the server user profile and password are placed into server configuration files, which should be maintained in a secure way using operating system security. Additionally, you can password protect some WebSphere Application Server resources. These passwords are also placed in server configuration files. The server automatically encodes passwords to deter casual observation, but password encoding alone is not sufficient protection.

The following files are located in the /config subdirectory of your profile and they can contain user identifiers and passwords:

- cells/*cell_name*/security.xml
- cells/*cell_name*/nodes/*node_name*/resources.xml
- cells/*cell_name*/nodes/*node_name*/servers/*server_name*/server.xml

For example, for the default profile, the *server_name* is server1.

The server user profile and password are used for authenticating the server when it initializes. This authentication is required for the following reasons:

- The user ID and password are used as the system identity for the server when an enterprise bean security is deployed to use SYSTEM_IDENTITY for method delegation. In this case, the user ID and password are used when method calls are made from one enterprise bean to another.

- The user ID and password are used to authenticate servers for inter-server communication. Because security for these files can be compromised, use a non-default user profile for the server identity and password. The default user profile is QEJBSVR. If you use the local OS user registry, you might choose to create and use a user profile that has no special authorities. For more information, see Running application servers under specific user profiles.

Secure user profiles for WebSphere Application Server

When WebSphere Application Server is first installed, by default, it uses the following user profiles:

QEJB This profile provides access to some administrative data, including passwords.

QEJBSVR
This profile provides the context in which your WebSphere Application Server runs. For security or administrative purposes, you might want to create other user profiles under which to run various parts of WebSphere Application Server. For more information, see Running application servers under specific user profiles.

Encoding password in files:

Use the **PropFilePasswordEncoder** utility to encode your passwords in the files. WebSphere Application Server does not provide a utility for decoding the passwords.

WebSphere Application Server contains several encoded passwords that are not encrypted. WebSphere Application Server provides the **PropFilePasswordEncoder** utility, which you can use to encode these passwords. However, the utility does not encode passwords that are contained within XML or XMI files. Instead, WebSphere Application Server automatically encodes the passwords in the following XML or XMI files.

Table 3. XML and XMI files that contain encoded passwords

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

Table 3. XML and XMI files that contain encoded passwords (continued)

File name	Additional information
profile_root/config/cells/*cell_name*/nodes/*node_name*/servers/security.xml	The following fields contain encoded passwords: • Keystore password • Truststore password • Cryptographic token device password • Session persistence password • DRS client data replication password
profile_root/config/cells/*cell_name*/nodes/*node_name*/servers/server1/resources.xml	The following fields contain encoded passwords: • WAS40Datasource password • mailTransport password • mailStore password • MQQueue queue mgr password
ibm-webservices-bnd.xmi	
ibm-webservicesclient-bnd.xmi	

You can use the **PropFilePasswordEncoder** utility to encode the passwords that are located in the following files.

Table 4. Files that you can encode using the PropFilePasswordEncoder utility

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

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

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

   V6.0.x If you are encoding the SAS properties files again, type: PropFilePasswordEncoder *"file_name"* -sas and the PropFilePasswordEncoder file encodes the known SAS properties.

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

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

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

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

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

Manually encoding passwords in properties files:

To use password encoding with WebSphere Application Server administrative commands and Java clients, passwords must be manually encoded in the `soap.client.props` and `sas.client.props` files using the **PropFilePasswordEncoder** tool.

To run the script, your user profile must have *ALLOBJ authority.

Use the **PropFilePasswordEncoder** utility to encode the passwords in properties files. The **PropFilePasswordEncoder** utility is a Qshell script. Complete the following steps to manually encode the passwords:

1. Sign on the server with a user profile that has all object (*ALLOBJ) special authority.
2. Run the Start Qshell (STRQSH) command on a command line to start the Qshell environment.
3. Use the **PropFilePasswordEncoder** utility to encode the passwords.

   For example, to encode the passwords for properties in the `sas.client.props` file for the default WebSphere Application Server profile (in a default installation), enter the following command:

   ```
   profile_root/bin/PropFilePasswordEncoder
    -profileName server1
    profile_root/properties/sas.client.props -SAS
   ```

   For example, to encode the passwords for properties in the `soap.client.props` file for the default standalone application server profile, enter the following command:

   ```
   profile_root/bin/PropFilePasswordEncoder
    -profileName server1
    profile_root/properties/soap.client.props
    com.ibm.SOAP.loginPassword,com.ibm.ssl.keyStorePassword,
    com.ibm.ssl.trustStorePassword
   ```

   For more information on the `sas.client.props` utility, see the "PropFilePasswordEncoder command reference."

The passwords are encoded in the `soap.client.props` and `sas.client.props` files.

See "Restoring or replacing damaged validation list objects" on page 85 for information on how to restore or replace a damaged validation list object.

PropFilePasswordEncoder command reference:

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

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

- V6.0.x *app_server_root*/bin

To run this script, your user profile must have *ALLOBJ authority.

Important: SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

Syntax

V6.0.x The command syntax is as follows:

```
PropFilePasswordEncoder "fileName" { passwordPropertiesList
  | -SAS } [ -profileName profile ] [ -help | -? ]
```

Important: You must specify either the passwordPropertiesList parameter or the -SAS parameter.

Parameters

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

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

passwordPropertiesList
This parameter is required if you are encoding passwords in the soap.client.props file. Specify one or more password properties that you want to encode.

V6.0.x **-SAS**
This parameter is required if you are encoding passwords in the sas.client.props file.

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

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

V6.0.x The following command encodes the passwords in the sas.client.props file for the default stand-alone application server profile:

```
app_server_root/bin/PropFilePasswordEncoder
 profile_root/default/properties/sas.client.props -SAS
```

The following command encodes the passwords in the soap.client.props file for the default stand-alone application server profile:

```
app_server_root/bin/PropFilePasswordEncoder
 profile_root/default/properties/soap.client.props
com.ibm.SOAP.loginPassword,com.ibm.ssl.keyStorePassword,com.ibm.ssl.trustStorePassword
```

Attention: These commands are displayed on multiple lines for illustrative purposes only.

Enabling the non-default OS/400 password encoding algorithm:

The purpose of password encoding is to deter casual observation of passwords in server configuration and property files.

Make sure all server instances within the administration console reside on the same i5/OS system.

By default, passwords are automatically encoded with a simple masking algorithm in various ASCII configuration files for WebSphere Application Server. You can manually encode passwords in properties files that are used by Java clients and by Application Server administrative commands.

For a description of the OS400 encoding algorithm, see “Password encoding and encryption” on page 77. To enable the OS400 password encoding algorithm for a WebSphere Application Server profile, complete these steps:

1. Set the os400.security.password properties to turn on the OS400 password encoding algorithm and to specify which the validation list object to use.

   Use the same validation list object for all WebSphere Application Server profiles. However, it is not recommended if you do not back up the objects and data for all profiles simultaneously. Consider your backup and restore policy when you decide what validation list object to use for each WebSphere Application Server profile.

   To set the properties, complete one of these steps:

   - Use the **-os400passwords** and **-validationlist** options for the manageprofiles -create utility, which is located in the *app_server_root*/bin directory, to set the properties when creating the profile. To create a WebSphere Application Server profile named prod, and to enable that profile for the OS400 encoding algorithm using the /QSYS.LIB/QUSRSYS.LIB/WAS.VLDL validation list object, you can complete the following steps:
     a. Run the Start Qshell (STRQSH) command on the i5/OS command line.
     b. In Qshell, run the following command:

        ```
        app_server_root/bin/manageprofiles
        -create -profileName prod -startingPort 10150
        -templatePath default -os400passwords
        -validationlist /QSYS.LIB/QUSRSYS.LIB/WAS.VLDL
        ```

        The previous command is on multiple lines for illustration purposes only.
   - Set the Java system properties in the **setupCmdLine** Qshell script of the WebSphere Application Server profile. To enable the OS400 password encoding algorithm, edit the *profile_root*/bin/setupCmdLine script using the following steps:
     a. Set the os400.security.password.encoding.algorithm property to OS400. The default setting is XOR.
     b. Set the os400.security.password.validation.list.object property to the absolute name of the validation list that you need to use. The default setting is /QSYS.LIB/QUSRSYS.LIB/EJSADMIN.VLDL.
     c. Save the file.
2. Grant the QEJB user profile run authority (*X) to the library that contains the validation list. If QEJB already has the minimum required authority (*X) to access the library, then proceed to the next step.
   a. Use the Display Authority (DSPAUT) to check for the minimum required authority if the validation list is created in the /QSYS.LIB/WSADMIN.LIB file.

      For example:

      ```
      DSPAUT OBJ('/QSYS.LIB/WSADMIN.LIB')
      ```
   b. Use the Change Authority (CHGAUT) command to grant run authority to the QEJB profile only if the QEJB profile does not already have this authority.

      For example:

      ```
      CHGAUT OBJ('/QSYS.LIB/WSADMIN.LIB') USER(QEJB) DTAAUT(*X)
      ```
3. Create a native validation list object (*VLDL). This step is optional for server profiles. The validation list object is created when the server is started. For remote profiles, create the validation list if the validation list does not already exist on the system that hosts the remote profile. Also, consider your backup and restore policy when you decide what validation list object to use with each remote profile.

   Attention: When you use the OS400 password encoding algorithm, the Java client is not required to reside on the same i5/OS system as the WebSphere Application Server profile that the client accesses.

To create a validation list object, perform the following steps with an i5/OS user profile that has *ALLOBJ special authority:

a. Sign on the server with a user profile that has the *ALLOBJ special authority.
b. Use the Create Validation List (**CRTVLDL**) command to create the validation list object.

   For example, to create the WSVLIST validation list object in the WSADMIN.LIB library, use the following command:

   ```
   CRTVLDL VLDL(WSADMIN/WSVLIST)
   ```

c. Grant the QEJB user profile *RWX authority to the validation list object. For example, to grant *RWX authority to the WSVLIST validation list object in the WSADMIN library , use the following command:

   ```
   CHGAUT OBJ('/QSYS.LIB/WSADMIN.LIB/WSVLIST.VLDL') USER(QEJB) DTAAUT(*RWX)
   ```

4. Use the Change System Value (**CHGSYSVAL**) command to set the QRETSVRSEC system value to 1. For example:

   ```
   CHGSYSVAL SYSVAL(QRETSVRSEC) VALUE('1')
   ```

5. For server profile, start or restart the server and wait until the server is ready for service before attempting to manually encode passwords in properties files that belong to the profile.

You have enabled the OS400 password encoding algorithm.

After completing the previous steps and restarting the server, you can manually encode passwords in properties files. See “Manually encoding passwords in properties files” on page 82 for more information.

Using the non-default OS/400 password encoding algorithm:

Use these steps to change your encoding algorithm from OS400 to XOR.

WebSphere Application Server supports both the OS400 and the XOR encoding algorithms. The default encoding algorithm is XOR. For more conceptual information on these algorithms, see “Password encoding and encryption” on page 77.

There might be instances where you need to use the OS400 encoding algorithm. See “Enabling the non-default OS/400 password encoding algorithm” on page 83.

If you use the OS400 encoding algorithm, but change the os400.security.password.encoding.algorithm property value to `XOR` as described in the first step, then all of the passwords remain encoded with the OS400 algorithm. However, after you restart the server, when you make changes to the passwords through the administrative console, the passwords are encoded using the XOR algorithm.

The following steps describe how to change your encoding algorithm from OS400 to XOR:

1. Set the os400.security.password.encoding.algorithm property to `XOR` for each WebSphere Application Server profile that uses the Validation list object.
2. Stop all of the servers.
3. Edit the configuration files and change all of the encoded passwords to their unencoded values.
4. Edit the properties files and change all of the encoded passwords to their clear text values.

You have changed your encoding algorithm from OS400 to XOR.

After you completing these steps, you can encode the passwords in the properties files. See “Manually encoding passwords in properties files” on page 82 for more information.

Restoring or replacing damaged validation list objects:

Periodically, you should save your validation list objects with the other configuration data objects that are used by WebSphere Application Server. Use this task if you need to restore or replace a damaged validation list object.

You can share validation lists between multiple WebSphere Application Server profiles. For example, if you have two profiles of WebSphere Application Server, default and prod, both profiles can use the `/QSYS.LIB/QUSRSYS.LIB/EJSADMIN.VLDL` validation list.

To restore or replace a damaged validation list object, complete the following steps:

1. Replace the encoded passwords with the unencoded value of the password for all of the WebSphere Application Server profiles that use the validation list object. To replace the password values, complete the following steps:
   a. Stop each of the servers.
   b. Set the os400.security.password.validation.list.object property for all of the servers to the absolute name of the new validation list that you want to use. You can use an existing validation list object or specify a new object. For new validation list objects, create them manually or use the objects that are created automatically when the server is restarted. For more information on manually creating validation list objects, see “Manually encoding passwords in properties files” on page 82.
   c. Edit the configuration files and set each encoded password to the appropriate clear text value.
2. Edit the `sas.client.props` and `soap.client.props` files and set each encoded password to the appropriate unencoded value before manually encoding the passwords.
3. Restart the servers for all of the WebSphere Application Server profiles whose validation list objects that are replaced.

After restarting the server, you have successfully replaced a damaged validation list object.

For additional information on backing up your data objects, see “Backing up security configuration files” on page 88.

Enabling custom password encryption:

After creating the server profile, perform this task to better protect passwords contained in configuration.

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

Complete the following steps to enable custom password encryption.

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

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

   Tip: If the custom encryption class name is com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl, it is automatically enabled when this class is present in the classpath. Do not define the system properties that are listed previously when the custom implementation has this package and class name. To disable encryption for this class, you must specify `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false` as a system property.
2. Add the Java archive (JAR) file containing the implementation class to the *app_server_root*`/classes` directory so that the WebSphere Application Server runtime can load the file.
3. Restart all server processes.

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

Custom password encryption is enabled.

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

Disabling custom password encryption:

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

Enable custom password encryption.

Complete the following steps to disable custom password encryption.

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

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

Custom password encryption is disabled.

Backing up security configuration files:

Back up your security configuration files to prevent the loss of information due to a potential system failure.

Consider backing up the following security information:

- Back up your user profiles.

  When you use local OS security, back up your user profiles, using the normal save procedures for user profiles. For more information, see the Backup and Recovery Guide by searching for ″Saving group and user profiles″ in the iSeries information center.

  For information about the Directory Services Product (LDAP server), see the iSeries information center.

  For information about Lotus Domino, see the Lotus Domino reference library.

- Back up your security property files.

  Security settings are saved in several properties files. By default, these properties are located in the*profile_root*/properties directory. The default standalone profile name is default. If you define additional WebSphere Application Server profiles, there are additional properties files located in the directories for those profiles.

  The following command saves all of the properties in the /SAS subdirectory:

  ```
  SAV DEV('/QSYS.lib/wsalib.lib/wsasavf.file')
  OBJ(('profile/properties/sas*'))
  ```

  This previous command is on two lines for illustrative purposes only. Enter it as one continuous line

  You can save security property files while WebSphere Application Server is running.

- Back up your HTTP configuration.

  The following information applies to IBM HTTP Server. If you are using Lotus Domino HTTP Server, see the Notes.net Documentation Library.

  Changes to the HTTP configuration are often made to enable WebSphere Application Server to serve servlets and JavaServer Pages (JSP) file requests and to enable WebSphere Application Server security. Consider saving your HTTP configuration as a part of your WebSphere Application Server backup and recovery. The IBM HTTP Server configurations are stored as members of the QATMHTTPC file in the QUSRSYS library. HTTP server instances are members of the QATMHINSTC file in the QUSRSYS library. The following example commands back up these files:

  ```
  SAVOBJ OBJ(QUSRSYS/QATMHTTPC)

  SAVOBJ OBJ(QUSRSYS/QATMHINSTC)
  ```

- Back up your key files.

  The key files contain certificates that are used by the security infrastructure for WebSphere Application Server. These certificates are also used for HTTPS transport between servers. Save all of the files in the WAS_INSTANCE_ROOT/etc directory. Key files are contained in the WAS_INSTANCE_ROOT/etc directory, but administrators might create and store these files in other directories.

- Back up your validation lists.

  Passwords are stored as encrypted data in validation list objects when you use the OS/400 password encoding algorithm. The default validation list is /QSYS.LIB/QUSRSYS.LIB/EJSADMIN.VLDL, but you can change it in the administrative console by specifying it as a system property for the application server. For more information, see Administering application servers.

For more information on backup and recovery strategies, see the following references:

- iSeries Information Center
- iSeries Backup and Recovery Version 5
- Security: Resources for learning

SPNEGO trust association interceptor (TAI) troubleshooting tips

Presented here is a list of trouble shooting tips useful in diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) TAI problems and exceptions.

The IBM Java Generic Security Service (JGSS) and IBM SPNEGO providers use a Java virtual machine (JVM) custom property to control trace information. The SPNEGO TAI uses the JRas facility to allow an administrator to trace only specific classes. To debug the TAI using tracing, the following important trace specifications or JVM customer should be used:

Table 5. SPNEGO TAI trace specifications

Trace	Use
com.ibm.security.jgss.debug	Set this **JVM Custom Property** to `all` to trace through JGSS code. Messages appear in the `trace.log` file, and **SystemOut.log**.
com.ibm.security.krb5.Krb5Debug	Set this **JVM Custom Property** to `all` to trace through the Kerberos5-specific JGSS code. Messages appear in the `trace.log` file, and **SystemOut.log**.
*com.ibm.ws.security.spnego.**	Set this trace on using the **administrative console > troubleshooting > Logging and Tracing > server1 > Change Log Detail Levels > com.ibm.ws.security.spnego.***. Messages appear in the `trace.log` file.

Problem: WebSphere Application Server and the Active Directory (AD) Domain Controller's time are not synchronized within 5 minutes:

The time is not synchronized between WebSphere Application Server and AD Domain Controller.

```
[2/24/06 13:12:46:093 CST] 00000060 Context       2 com.ibm.ws.security.spnego.Context
    begin GSSContext accepted
[2/24/06 13:12:46:093 CST] 00000060 Context       E com.ibm.ws.security.spnego.Context
    begin
CWSPN0011E: An invalid SPNEGO token has been encountered while authenticating a
    HttpServletRequest:
0000:  60820160 06062b06 01050502 a1820154    `..` ..+. .... ...T
0010:  30820150 a0030a01 01a10b06 092a8648    0..P .... .... .*.H
0020:  82f71201 0202a282 013a0482 01366082    .... .... .:.. .6`.
0030:  01320609 2a864886 f7120102 0203007e    .2.. *.H. .... ...~
0040:  82012130 82011da0 03020105 a1030201    ..!0 .... .... ....
0050:  1ea41118 0f323030 36303232 34313931    .... .200 6022 4191
0060:  3234365a a5050203 016b48a6 03020125    246Z .... .kH. ...%
0070:  a9161b14 57535345 432e4155 5354494e    .... WSSE C.AU STIN
0080:  2e49424d 2e434f4d aa2d302b a0030201    .IBM .COM .-0+ ....
0090:  00a12430 221b0448 5454501b 1a773230    ..$0 "..H TTP. .w20
00a0:  30337365 63646576 2e617573 74696e2e    03se cdev .aus tin.
00b0:  69626d2e 636f6dab 81aa1b81 a76f7267    ibm. com. .... .org
00c0:  2e696574 662e6a67 73732e47 53534578    .iet f.jg ss.G SSEx
00d0:  63657074 696f6e2c 206d616a 6f722063    cept ion,  maj or c
00e0:  6f64653a 2031302c 206d696e 6f722063    ode:  10,  min or c
00f0:  6f64653a 2033370a 096d616a 6f722073    ode:  37. .maj or s
0100:  7472696e 673a2044 65666563 74697665    trin g: D efec tive
0110:  20746f6b 656e0a09 6d696e6f 72207374     tok en.. mino r st
0120:  72696e67 3a20436c 69656e74 2074696d    ring : Cl ient  tim
0130:  65204672 69646179 2c204665 62727561    e Fr iday , Fe brua
0140:  72792032 342c2032 30303620 61742031    ry 2 4, 2 006  at 1
0150:  3a31323a 34352050 4d20746f 6f20736b    :12: 45 P M to o sk
0160:  65776564                               ewed
```

Solution: You can fix this in one of two ways. The preferred way is to synchronize the WebSphere Application Server system time to within 5 minutes of the AD server's time. A best practice is to use a time server to keep all systems synchronized. Or you can add or adjust the clockskew parameter in the Kerberos configuration file.

Note: The default for the clockskew parameter is 300 seconds (or 5 minutes).

Problem: Getting exception: No factory available to create a name for mechanism 1.3.6.1.5.5.2:

There apparently is no factory available to process the creation of a name for the specific mechanism.

The `systemout.log` file displays something like this:

```
[4/8/05 22:51:24:542 EDT] 5003e481 SystemOut     O [JGSS_DBG_PROV] Provider
        IBMJGSSProvider version 1.01 does not support mech 1.3.6.1.5.5.2
[4/8/05 22:51:24:582 EDT] 5003e481 ServerCredent >
        com.ibm.ws.security.spnego.ServerCredential initialize ENTRY
SPNEGO014: Kerberos initialization Failure: org.ietf.jgss.GSSException, major code: 2,
        minor code: 0
 major string: Unsupported mechanism
 minor string: No factory available to create name for mechanism 1.3.6.1.5.5.2
 at com.ibm.security.jgss.i18n.I18NException.throwGSSException
        (I18NException.java:30)
 at com.ibm.security.jgss.GSSManagerImpl.a(GSSManagerImpl.java:36)
 at com.ibm.security.jgss.GSSCredentialImpl.add(GSSCredentialImpl.java:217)
 at com.ibm.security.jgss.GSSCredentialImpl.<init>(GSSCredentialImpl.java:264)
```

Solution: Check the `java.security` file to ensure it contains the IBMSPNEGO security provider and that the provider is defined correctly. The `java.security` file should contain a line similar to:

```
security.provider.6=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
```

Problem: Getting an exception:

An exception has occurred when reporting to the client.

You get the following display.

```
Error authenticating request. Reporting to client
Major code = 11, Minor code = 31
org.ietf.jgss.GSSException, major code: 11, minor code: 31
 major string: General failure, unspecified at GSSAPI level
 minor string: Kerberos error while decoding and verifying token:
                com.ibm.security.krb5.internal.KrbException, status code: 31
 message: Integrity check on decrypted field failed
```

as the JGSS library is trying to process the SPNEGO token.

Cause: This exception is the result of encoding the ticket using one key and attempting to decode it using a different key. There are number of possible reasons for this condition:

1. The Kerberos keytab file has not been copied to the server machine after it has been regenerated.
2. The Kerberos configuration points to the wrong Kerberos keytab file.
3. The Kerberos service principal name (SPN) has been defined to the Active Directory more than once; this can occur because you have another userid with a similarly defined SPN (either exactly the same name, or one having a different name but with a port defined part of the SPN).

Solution: If the problem is with the Kerberos keytab file, then fix it. If the problem is with multiple SPN definitions, then remove the extra or conflicting SPN, confirm that the SPN is no longer registered with the Active Directory, and then add the SPN. The Active Directory may need to be searched for other entries with SPNs defined that clash with the SPN.

To confirm that the SPN is not registered, the command:

```
setspn –l userid
```

should return with the following response:

```
Cannot find account userid
```

Problem: Single sign-on is not occurring.:

When trace is turned on, the following message appears:

```
[2/27/06 14:28:04:191 CST] 00000059 SpnegoHandler <
         com.ibm.ws.security.spnego.SpnegoHandler handleRequest: Received a
         non-SPNEGO Authorization Header RETURN
```

Cause: The client is returning an NT LAN manager (NTLM) response to the authorize challenge, not a SPNEGO token. This condition can be occur due to any of the following reasons:

- The client has not been configured properly.
- The client is not using a supported browser. For example, when using Microsoft Internet Explorer 5.5, SP1 responds with a non-SPNEGO authentication header.
- The user has not logged into the Active Directory domain, or into a trusted domain, or the client used does not support integrated authentication with Windows – in this case, the SPNEGO TAI is working properly.
- The user is accessing a service defined on the same machine upon which the client is running (local host). Microsoft Internet Explorer resolves the host name of the URL to `http://localhostsomeURL` instead of a fully qualified name.
- The SPN is not found in the Active Directory. The SPN must be of the format `HTTP/server.realm.com`. The command to add the SPN is

  ```
  setspn –a HTTP/server.realm.com userid
  ```

If the SPN is defined incorrectly as `HTTP/server.realm.com@REALM.COM` with the addition of `@REALM.COM`, then delete the user, redefine the user, and redefine the SPN.

Problem: Credential Delegation is not working:

An invalid option is detected. When trace is turned on, the following message is displayed:

```
com.ibm.security.krb5.KrbException, status code: 101 message: Invalid option in
ticket request
```

Cause: The Kerberos configuration file is not properly configured.

Solution: Ensure that neither renewable, nor proxiable are set to true.

Problem: Unable to get SSO working using RC4-HMAC encryption.:

When trace is turned on, you get the following message in the trace:

```
com.ibm.security.krb5.internal.crypto.KrbCryptoException, status code: 0
 message: Checksum error; received checksum does not match computed checksum
```

Cause: RC4-HMAC encryption is not supported with a Microsoft Windows 2000 Kerberos key distribution center (KDC). To confirm this condition, examine the trace and identify where the exception is thrown. The content of the incoming ticket should be visible in the trace. Although the incoming ticket is encrypted, the SPN for the service is readable. If a Microsoft Windows 2000 KDC is used and the system is configured to use RC4-HMAC, the string representing the ticket for `userid@REALM` (instead of the expected `HTTP/hostname.realm@REALM`) is displayed. For example, this is beginning of the ticket received from a Microsoft Windows 2000 KDC:

```
0000: 01 00 6e 82 04 7f 30 82  04 7b a0 03 02 01 05 a1  ..n...0.....
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  .............
0020: a5 61 82 03 a1 30 82 03  9d a0 03 02 01 05 a1 0a  .a...0.....
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 18 30 16 a0 03  ...REALM.COM.0..
0040: 02 01 01 a1 0f 30 0d 1b  0b 65 70 66 64 77 61 73  .....0...userid
0050: 75 6e 69 74 a3 82 03 6e  30 82 03 6a a0 03 02 01  .a.f...n0..j....
```

The realm is REALM.COM. The service name is userid. A correctly formed ticket for the same SPN is:

```
0000: 01 00 6e 82 04 56 30 82  04 52 a0 03 02 01 05 a1  ..n..V0..R.....
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  .............
0020: 82 61 82 03 7e 30 82 03  7a a0 03 02 01 05 a1 0a  .a...0..z.....
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 2a 30 28 a0 03  ..REALM.COM.0...
0040: 02 01 02 a1 21 30 1f 1b  04 48 54 54 50 1b 17 75  .....0...HTTP..u
0050: 73 31 30 6b 65 70 66 77  61 73 73 30 31 2e 65 70  serid.realm.com.
0060: 66 64 2e 6e 65 74 a3 82  03 39 30 82 03 35 a0 03  ...n.....90..5..
```

Solution: To correct the problem, either use the Single data encryption standard (DES) or use a Microsoft Windows 2003 Server for a KDC. Remember to regenerate the SPN, and the Kerberos keytab file.

Problem: User receives the following message when accessing a protected URL through the SPNEGO SSO:

```
Bad Request

Your browser sent a request that this server could not understand.
Size of request header field exceeds server limit.

Authorization: Negotiate YII......
```

Cause: This message is generated by the Apache/IBM HTTP Server. This server is indicating that the authorization header returned by the user's browser is too large. The long string that follows the word Negotiate (in the error message above) is the SPNEGO token. This SPNEGO token is a wrapper of the Microsoft Windows Kerberos token. Microsoft Windows includes the user's PAC information in the Kerberos token. The more security groups that the user belongs to, the more PAC information is inserted in the Kerberos token, and the larger the SPNEGO becomes. IBM HTTP Server 2.0 (also Apache 2.0 and IBM HTTP Server 6.0) limit the size of any acceptable HTTP header to be 8K. In Microsoft Windows domains having many groups, and with user membership in many groups, the size of the user's SPNEGO token may exceed the 8K limit.

Solution: If possible, reduce the number of security groups the user is a member of. IBM HTTP Server 2.0.47 cumulative fix PK01070 allows for HTTP header sizes up to and beyond the Microsoft limit of 12K. WebSphere Application Server Version 6.0 users can obtain this fix in fixpack 6.0.0.2.

Note: Non-Apache based Web servers may require differing solutions.

Problem: Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log:

Cause: While most of the JGSS tracing is controlled by the com.ibm.security.jgss.debug property, a small set of messages are controlled by the com.ibm.security.krb5.Krb5Debug property. The com.ibm.security.krb5.Krb5Debug property has a default value to put some messages to the **SystemOut.log**.

Solution: To remove all KRB_DBG_KDC messages from the **SystemOut.log**, set the JVM property as follows:

```
-Dcom.ibm.security.krb5.Krb5Debug=none
```

.

Problem: HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login.:

Cause: The Microsoft Internet Explorer maintains state during a user's request. If a request was given the response of an ″HTTP 401 Authenticate Negotiate″, and the browser responds with a NTLM token obtained through a userid/password challenge, the browser resubmits the request. If this second request is given a response of an HTML page containing a redirection to the same URL but with new arguments (via Javascript) then the browser does not resubmit the POST parameters. To avoid this problem, it is critical to NOT perform the automatic redirection. If the user clicks on a link, the problem does not occur. See section 5.2 Client Returns NTLM Token to SPNEGO Challenge for a resolution to the problem,

Solution: The browser responds to the Authenticate/Negotiate challenge with an NTLM token, not an SPNEGO token. The SPNEGO TAI sees the NTLM, and returns back a HTTP 403 response, along with the HTML page. When the browser runs the Javascript redirTimer function, any POST of GET parameters that were present on the original request are lost.

By leveraging the SPN<id>.NTLMTokenReceivedPage property, an appropriate message page can be returned to the user. The default message that is returned (in the absence of a user defined property) is:

```
"<html><head><title>An NTLM Token was Received.</title></head>"
 + "<body>Your browser configuration is correct, but you have not logged into
          a supported Windows Domain."
 + "<p>Please login to the application using the normal login page.</html>";
```

Using the SPN<id>.NTLMTokenReceivedPage property, you can customize the exact response. It is critical that the returned HTML not perform a redirection.

When the SPNEGO TAI has been configured to use the shipped default HTTPHeaderFilter class as the SPN<id>.filterClass, then the SPN<id>.filter can be used to allow the second request to flow directly to the normal WebSphere Application Server security mechanism. In this way, the user experiences the normal authentication mechanism.

An example of such a configuration follows. The required SPNEGO TAI properties necessary and the HTML file content are presented.

Table 6. SPNEGO TAI properties and HTML

SPNEGO TAI Property Name	HTML File Content
com.ibm.ws.security.spnego.SPN1.hostName	server.wasteched30.torolab.ibm.com
com.ibm.ws.security.spnego.SPN1.filterClass	com.ibm.ws.security.spnego.HTTPHeaderFilter
com.ibm.ws.security.spnego.SPN1.filter	request-url!=noSPNEGO
com.ibm.ws.security.spnego.SPN1.NTLMTokenReceivedPage	File:///C:/temp/NTLM.html

Note: Observe that the filter property instructs the SPNEGO TAI to NOT intercept any HTTP request that contains the string “noSPNEGO”.

Here is an example of a generating a helpful response.

```
<html>
<head>
<title>NTLM Authentication Received </title>
<script language="javascript">
 var purl=""+document.location;
if (purl.indexOf("noSPNEGO")<0) {
  if(purl.indexOf('?')>=0) purl+="&noSPNEGO";
  else purl+="?noSPNEGO";
 }
</script>
```

```
</head>
<body>
<p>An NTLM token was retrieved in response to the SPNEGO challenge. It is likely that
you are not logged into a Windows domain.<br>
Click on the following link to get the requested website.
<script language="javascript">
 document.write("<a href='"+purl+"'>");
 document.write("Open the same page using the normal authentication
  mechanism.");
 document.write("</a><br>");
</script>
You will not automatically be redirected.
</body>
</html>
```

Object Request Broker

Object Request Broker tuning guidelines

Use the guidelines in this document any time the Object Request Broker (ORB) is used in a workload.

The ORB is used whenever enterprise beans are accessed through a remote interface. If you experience particularly high or low CPU consumption, you might have a problem with the value of one of the following parameters. Examine these core tuning parameters for every application deployment.

Thread pool adjustments:
Size

Tune the size of the ORB thread pool according to your workload. Avoid suspending threads because they have no work ready to process. If threads do not have work ready to process, CPU time is consumed by calling the Object.wait method, performing a context switch. Tune the thread pool size such that the length of time that the threads wait is short enough to prevent them from being destroyed because they are idle too long.

The thread pool size is dependent on your workload and system. In typical configurations, applications need 10 or fewer threads per processor.

However, if your application is performing a very slow backend request, like a request to a database system, a server thread blocks waiting for the backend request to complete. With backend requests, CPU use is fairly low. In this case, increasing the load does not increase CPU use or throughput. Your thread dumps indicate that nearly all the threads are in a call out to the backend resource. In this case, consider increasing the number of threads per processor until throughput improves and thread dumps show that the threads are in other areas of the run time besides the backend call. You should adjust the number of threads only if your backend resource is tuned correctly.

The **Allow thread allocation beyond maximum thread size** parameter also affects thread pool size, but do not use this parameter unless your back end stops for long periods of time, causing the blocking of all the run-time threads waiting for the backend system instead of processing other work that does not involve the backend system.

You can adjust the thread pool size settings in the administrative console. Click **Servers > Application servers >** *server_name* **> Container services > ORB service > Thread pool**. You can adjust the minimum and maximum number of threads. See Thread pool settings for more information.

Pass by reference:

Specifies how the ORB passes parameters. If enabled, the ORB passes parameters by reference instead of by value, to avoid making an object copy. If you do not enable the pass by reference option, a copy of the parameter passes rather than the parameter object itself. This can be expensive because the ORB must first make a copy of each parameter object.

You can use this option only when the Enterprise JavaBeans (EJB) client and the EJB are on the same classloader. This requirement means that the EJB client and the EJB must be deployed in the same EAR file.

If the Enterprise JavaBeans (EJB) client and server are installed in the same WebSphere Application Server instance, and the client and server use remote interfaces, enabling the pass by reference option can improve performance up to 50%. The pass by reference option helps performance only where non-primitive object types are passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

Important: Enable this property with caution because unexpected behavior can occur. If an object reference is modified by the callee, the caller's object is modified as well, since they are the same object.

If you use command-line scripting, the full name of this system property is com.ibm.CORBA.iiop.noLocalCopies.

Data type	Boolean
Default	Not enabled (false)

The use of this option for enterprise beans with remote interfaces violates Enterprise JavaBeans (EJB) Specification, Version 2.0 (see section 5.4). Object references passed to Enterprise JavaBeans (EJB) methods or to EJB home methods are not copied and can be subject to corruption.

Consider the following example:

```
Iterator iterator = collection.iterator();
MyPrimaryKey pk = new MyPrimaryKey();
while (iterator.hasNext()) {
   pk.id = (String) iterator.next();
   MyEJB myEJB = myEJBHome.findByPrimaryKey(pk);
}
```

In this example, a reference to the same MyPrimaryKey object passes into WebSphere Application Server with a different ID value each time. Running this code with pass by reference enabled causes a problem within the application server because multiple enterprise beans are referencing the same MyPrimaryKey object. To avoid this problem, set the com.ibm.websphere.ejbcontainer.allowPrimaryKeyMutation system property to `true` when the pass by reference option is enabled. Setting the pass by reference option to `true` causes the EJB container to make a local copy of the PrimaryKey object. As a result, however, a small portion of the performance advantage of setting the pass by reference option is lost.

As a general rule, any application code that passes an object reference as a parameter to an enterprise bean method or to an EJB home method must be scrutinized to determine if passing that object reference results in loss of data integrity or in other problems.

After examining your code, you can enable the pass by reference option by setting the com.ibm.CORBA.iiop.noLocalCopies system property to `true`. You can also enable the pass by reference option in the administrative console. Click **Servers** > **Application servers** > *server_name* > **Container services** > **ORB Service** and select **Pass by reference**.

Fragment size: The ORB separates messages into fragments to send over the ORB connection. You can configure this fragment size through the com.ibm.CORBA.FragmentSize parameter.

To determine and change the size of the messages that transfer over the ORB and the number of required fragments, perform the following steps:

1. In the administrative console, enable ORB tracing in the ORB Properties page. See Object Request Broker service settings for more information.
2. Enable ORBRas tracing from the logging and tracing page.
3. Increase the trace file sizes because tracing can generate a lot of data.
4. Restart the server and run at least one iteration (preferably several) of the case that you are measuring.
5. Look at the traceable file and do a search for Fragment to follow: Yes.

   This message indicates that the ORB transmitted a fragment, but it still has at least one remaining fragment to send before the entire message arrives. A Fragment to follow: No value indicates that the particular fragment is the last in the entire message. This fragment can also be the first, if the message fit entirely into one fragment.

   If you go to the spot where Fragment to follow: Yes is located, you find a block that looks similar to the following example:

   Fragment to follow: Yes
   Message size: 4988 (0x137C)
   --
   Request ID: 1411

   This example indicates that the amount of data in the fragment is 4988 bytes and the Request ID is 1411. If you search for all occurrences of Request ID: 1411, you can see the number of fragments that are used to send that particular message. If you add all the associated message sizes, you have the total size of the message that is being sent through the ORB.
6. You can configure the fragment size by setting the com.ibm.CORBA.FragmentSize property. See Object Request Broker custom properties for more information.

Interceptors: Interceptors are ORB extensions that can set up the context before the ORB runs a request. For example, the context might include transactions or activity sessions to import. If the client creates a transaction, and then flows the transaction context to the server, then the server imports the transaction context onto the server request through the interceptors.

Most clients do not start transactions or activity sessions, so most systems can benefit from removing the interceptors that are not required.

To remove the interceptors, manually edit the server.xml file and remove the interceptor lines that are not needed from the ORB section.

Connection Cache Adjustments: Depending on an application server's workload, and throughput or response-time requirements, you might need to adjust the size of the ORB's connection cache. Each entry in the connection cache is an object that represents a distinct TCP/IP socket endpoint, identified by the hostname or TCP/IP address, and the port number used by the ORB to send a GIOP request or a GIOP reply to the remote target endpoint. The purpose of the connection cache is to minimize the time required to establish a connection by reusing ORB connection objects for subsequent requests or replies. (The same TCP/IP socket is used for the request and corresponding reply.)

For each application server, the number of entries in the connection cache relates directly to the number of concurrent ORB connections. These connections consist of both the inbound requests made from remote clients and outbound requests made by the application server. When the server-side ORB receives a connection request, it uses an existing connection from an entry in the cache, or establishes a new connection and adds an entry for that connection to the cache.

The ORB Connection cache maximum and Connection cache minimum properties are used to control the maximum and minimum number of entries in the connection cache at a given time. When the number of entries reaches the value specified for the Connection cache maximum property, and a new connection is needed, the ORB creates the requested connection, adds an entry to the cache and searches for and attempts to remove up to five inactive connection entries from the cache. Because the new connection is added before inactive entries are removed, it is possible for the number of cache entries to temporarily exceed the value specified for the Connection cache maximum property.

An ORB connection is considered inactive if the TCP/IP socket stream is not in use and there are no GIOP replies pending for any requests made on that connection. As the application workload diminishes, the ORB closes the connections and removes the entries for these connections from the cache. The ORB continues to remove entries from the cache until the number of remaining entries is at or below the value specified for the Connection cache maximum property. The number of cache entries is never less then the value specified for the Connection cache minimum property, which must be at least five connections less than the value specified for the Connection cache maximum property.

Adjustments to the connection cache in the client-side ORB are usually not necessary because only a small number of connections are made on that side.

JNI Reader Threads: By default, the ORB uses a Java thread for processing each inbound connection request it receives. As the number of concurrent requests increases, the storage consumed by a large number of reader threads increases and can become a bottleneck in resource-constrained environments. Eventually, the number of Java threads created can cause out-of-memory exceptions if the number of concurrent requests exceeds the system's available resources.

To help address this potential problem, you can configure the ORB to use JNI reader threads where a finite number of reader threads, implemented using native OS threads instead of Java threads, are created during ORB initialization. JNI reader threads rely on the native OS TCP/IP asynchronous mechanism that enables a single native OS thread to handle I/O events from multiple sockets at the same time. The ORB manages the use of the JNI reader threads and assigns one of the available threads to handle the connection request, using a round-robin algorithm. Ordinarily, JNI reader threads should only be configured when using Java threads is too memory-intensive for your application environment.

The number of JNI reader threads you should allocate for an ORB depends on many factors and varies significantly from one environment to another, depending on available system resources and workload requirements. The following potential benefits might be achieved if you use JNI threads:

- Because a fixed number of threads is allocated, memory usage is reduced. This reduction provides significant benefit in environments with unusually large and sustained client-request workloads.
- The time needed to dynamically create and destroy Java threads is eliminated because a fixed number of JNI threads is created and allocated during ORB initialization.
- Each JNI thread can handle up to 1024 socket connections and interacts directly with the asynchronous I/O native OS mechanism, which might provide enhanced performance of network I/O processing.

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 a brief description of each WebSphere extension.

In addition, now your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find

the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java 2 Platform, Enterprise Edition (J2EE) module basis.

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

Application profiling

Application profiling performance considerations:

Application profiling enables assembly configuration techniques that improve your application run time, performance and scalability. You can configure tasks that identify incoming requests, identify access intents determining concurrency and other data access characteristics, and profiles that map the tasks to the access intents.

The capability to configure the application server can improve performance, efficiency and scalability, while reducing development and maintenance costs. The application profiling service has no tuning parameters, other than a checkbox for disabling the service if the service is not necessary. However, the overhead for the application profile service is small and should not be disabled, or unpredictable results can occur.

Access intents enable you to specify data access characteristics. The WebSphere runtime environment uses these hints to optimize the access to the data, by setting the appropriate isolation level and concurrency. Various access intent hints can be grouped together in an access intent policy.

In WebSphere Application Server, it is recommended that you configure bean level access intent for loading a given bean. Application profiling enables you to configure multiple access intent policies on the entity bean, if desired. Some callers can load a bean with the intent to read data, while others can load the bean for update. The capability to configure the application server can improve performance, efficiency, and scalability, while reducing development and maintenance costs.

Access intents enable the EJB container to be configured providing optimal performance based on the specific type of enterprise bean used. Various access intent hints can be specified declaratively at deployment time to indicate to WebSphere resources, such as the container and persistence manager, to provide the appropriate access intent services for every EJB request.

The application profiling service improves overall entity bean performance and throughput by fine tuning the run time behavior. The application profiling service enables EJB optimizations to be customized for multiple user access patterns without resorting to ″worst case″ choices, such as pessimistic update on a bean accessed with the findByPrimaryKey method, regardless of whether the client needs it for read or for an update.

Application profiling provides the capability to define the following hierarchy: **Container-Managed Tasks** > **Application Profiles** > **Access Intent Policies** > **Access Intent Overrides**. Container-managed tasks identify units of work (UOW) and are associated with a method or a set of methods. When a method associated with the task is invoked, the task name is propagated with the request. For example, a UOW refers to a unique path within the application that can correspond to a transaction or ActivitySession. The name of the task is assigned declaratively to a J2EE client or servlet, or to the method of an enterprise bean. The task name identifies the starting point of a call graph or subgraph; the task name flows from the starting point of the graph downstream on all subsequent IIOP requests, identifying each subsequent invocation along the graph as belonging to the configured task. As a best practice, wherever a UOW starts, for example, a transaction or an ActivitySession, assign a task to that starting point.

The application profile service associates the propagated tasks with access intent policies. When a bean is loaded and data is retrieved, the characteristics used for the retrieval of the data are dictated by the application profile. The application profile configures the access intent policy and the overrides that should be used to access data for a specific task.

Access intent policies determine how beans are loaded for specific tasks and how data is accessed during the transaction. The access intent policy is a named group of access intent hints. The hints can be used, depending on the characteristics of the database and resource manager. Various access intent hints applied to the data access operation govern data integrity. The general rule is, the more data integrity, the more overhead. More overhead causes lower throughput and the opportunity for simultaneous data access from multiple clients.

If specified, access intent overrides provide further configuration for the access intent policy.

Best practices

Application profiling is effective in a variety of different scenarios. The following are example situations where application profiling is useful

- **The same bean is loaded with different data access patterns**

  The same bean or set of beans can be reused across applications, but each of those applications has differing requirements for the bean or for beans within the invocation graph. One application can require that beans be loaded for update, while another application requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.
- **Different clients have different data access requirements**

  The same bean or set of beans can be used for different types of client requests. When those clients have different requirements for the bean, or for beans within the invocation graph, application profiling can be used to tailor the bean loading characteristics to the requirements of the client. One client can require beans be loaded for update, while another client requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.

Monitoring tools

You can use the Tivoli Performance Viewer, database and logs as monitoring tools.

You can use the Tivoli Performance Viewer to monitor various metrics associated with beans in an application profiling configuration. The following sections describe at a high level the Tivoli Performance Viewer metrics that reflect changes when access intents and application profiling are used:

- **Collection scope**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor this information to determine the difference between using the ActivitySession scope versus the transaction scope. For the transaction scope, depending on how the container transactions are defined, activates and passivates can be associated with method invocations. The application could use the ActivitySession scope to reduce the frequency of activates and passivates. For more information, see ″Using the ActivitySession service.″
- **Collection increment**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular findByPrimaryKey operation. For example, if the collection increment is set to 10, rather than the default 25, the *Num Activates* value shows 25 for the initial findByPrimaryKey, before any result set iterator runs. If the number of activates rarely exceeds the collection increment, consider reducing the collection increment setting.
- **Resource manager prefetch increment**

The resource manager prefetch increment is a hint acted upon by the database engine to depend upon the database. The Tivoli Performance Viewer does not have a metric available to show the effect of the resource manager prefetch increment setting.

- **Read ahead hint**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular request. If a read ahead association is not in use, the *Num Activates* value shows a lower initial number. If a read ahead association is in use, the *Num Activates* value represents the number of activates for the entire call graph.

Database tools are helpful in monitoring the different bean loading characteristics that introduce contention and concurrency issues. These issues can be solved by application profiling, or can be made worse by the misapplication of access intent policies.

Database tools are useful for monitoring locking and contention characteristics, such as locks, deadlocks and connections open. For example, for locks the DB2 Snapshot Monitor can show statistics for lock waits, lock time-outs and lock escalations. If excessive lock waits and time-outs are occurring, application profiling can define specific client tasks that require a more string level of locking, and other client tasks that do not require locking. Or, a different access intent policy with less restrictive locking could be applied. After applying this configuration change, the snapshot monitor shows less locking behavior. Refer to information about the database you are using on how to monitor for locking and contention.

The **application server logs** can be monitored for information about rollbacks, deadlocks, and other data access or transaction characteristics that can degrade performance or cause the application to fail.

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 `explictBufferLimitOnStop` to configure the com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop 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.explictBufferLimitOnStop) 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.

Important: Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property 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*.
- To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field if it is not already selected.
2. Under Performance Settings, select Balanced performance and balanced memory usage or Custom.
3. In the Disk cache cleanup frequency field, specify an appropriate length of time, in minutes.

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.

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.

Important: Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property 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*.

- To configure an object cache instance, click **Resources > Cache instances > Object cache instances >** *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field, if it is not already selected.
2. Under Disk Cache settings, select Limit disk cache size in entries, if it is not already selected.
3. In the Disk cache size field, specify the number of cache IDs that can be saved in memory for the dependency ID and template buffers.

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

Tune the delay offload function

Use these properties to tune the delay offload function for the disk cache.

Important: Setting these custom properties manually is deprecated for V6.1. You should use the administrative console to set these properties. The individual property descriptions include information on how to use the administrative console to set these properties.

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.explictBufferLimitOnStop

Use this custom property when the flush-to-disk-on-stop feature is enabled. When the server is stopping, offloads are limited to the value specified for this property, pending removal of entries in the explicit invalidation buffer. If this property is set to `0`, there is no limit to the number of offloads that can occur. Only positive integers are accepted as values for this property. If the number of entries in the explicit invalidation buffer is greater then the specified limit, all of the disk files for this specified cache instance are deleted after the server stops.

Important: You cannot use the administrative console to set this property.

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

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

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

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

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

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

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

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

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

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

Dynamic cache MBean statistics:

The dynamic cache service provides an MBean interface to access cache statistics.

Access cache statistics with the MBean interface, using JACL

- Obtain the MBean identifier with the **queryNames** command, for example:

  ```
  $AdminControl queryNames type=DynaCache,*  // Returns a list of the available dynamic cache MBeans
  ```

  Select your dynamic cache MBean and run the following command:

  ```
  set mbean <dynamic_cache_mbean>
  ```

- Retrieve the names of the available cache statistics:

  ```
  $AdminControl invoke $mbean getCacheStatisticNames
  ```

- Retrieve the names of the available cache instances:

  ```
  $AdminControl invoke $mbean getCacheInstanceNames
  ```

- Retrieve all of the available cache statistics for the base cache instance:

  ```
  $AdminControl invoke $mbean getAllCacheStatistics
  ```

- Retrieve all of the available cache statistics for the named cache instance:

  ```
  $AdminControl invoke $mbean getAllCacheStatistics "services/cache/servletInstance_4"
  ```

- Retrieve cache statistics that are specified by the names array for the base cache instance:

  ```
  $AdminControl invoke $mbean getCacheStatistics
   {"DiskCacheSizeInMB ObjectsReadFromDisk4000K RemoteObjectMisses"}
  ```

  Note: This command should all be entered on one line. It is broken here for printing purposes.

- Retrieve cache statistics that are specified by the names array for the named cache instance:

  ```
  $AdminControl invoke $mbean getCacheStatistics
   {services/cache/servletInstance_4 "ExplicitInvalidationsLocal CacheHits"}
  ```

  Note: This command should all be entered on one line. It is broken here for printing purposes.

Accessing dynamic cache PMI counters:

The dynamic cache statistics interface is defined as WSDynamicCacheStats under the `com\ibm\websphere\pmi\stat` package.

Dynamic cache statistics are structured as follows in the Performance Monitoring Infrastructure (PMI) tree:

```
__Dynamic Caching+
  |
  |__<Servlet: instance_1>
    |__Templates+
      |__<template_1>
      |__<template_2>
    |__Disk+
      |__<Disk Offload Enabled>
  |
  |__<Object: instance_2>
    |__Object Cache+
      |__<Counters>
+ indicates logical group
```

StatDescriptor locates and accesses particular statistics in the PMI tree. For example:

1. StatDescriptor to represent statistics for cache servlet: `instance_1 templates group template_1: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Servlet: instance1", WSDynamicCacheStats.TEMPLATE_GROUP, "template_1"});`

2. StatDescriptor to represent statistics for cache servlet: `instance_1 disk group Disk Offload Enabled: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Servlet: instance_1", WSDynamicCacheStats.DISK_GROUP, WSDynamicCacheStats.DISK_OFFLOAD_ENABLED});`
3. StatDescriptor to represent statistics for cache object: `instance2 object cache group Counters: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, "Object: instance_2", WSDynamicCacheStats.OBJECT_GROUP, WSDynamicCacheStats.OBJECT_COUNTERS});`

Important: Cache instance names are prepended with cache type ("Servlet: " or "Object: ").

Counter definitions for Servlet Cache

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. MaxInMemoryCache EntryCount	`WSDynamicCacheStats.NAME - "Servlet: instance_1"`	The maximum number of in-memory cache entries.	5.0 and later
WSDynamicCacheStats. InMemoryCache EntryCount	`WSDynamicCacheStats.NAME - "Servlet: instance_1"`	The current number of in-memory cache entries	5.0 and later
WSDynamicCacheStats. HitsIn MemoryCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of requests for cacheable objects that are served from memory.	5.0 and later
WSDynamicCacheStats. HitsOnDiskCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of requests for cacheable objects that are served from disk.	5.0 and later
WSDynamicCacheStats. ExplicitInvalidationCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of explicit invalidations.	5.0 and later
WSDynamicCacheStats. LruInvalidationCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1`	The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. instance.	5.0 and later
WSDynamicCacheStats. TimeoutInvalidationCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The number of cache entries that are removed from memory and disk because their timeout has expired.	5.0 and later
WSDynamicCacheStats. InMemoryAndDisk CacheEntryCount	`WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"`	The current number of used cache entries in memory and disk.	5.0 and later

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. RemoteHitCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain.	5.0 and later
WSDynamicCacheStats. MissCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that were not found in the cache.	5.0 and later
WSDynamicCacheStats. ClientRequestCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that are generated by applications running on this application server.	5.0 and later
WSDynamicCacheStats. DistributedRequestCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of requests for cacheable objects that are generated by cooperating caches in this replication domain.	5.0 and later
WSDynamicCacheStats. ExplicitMemory InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations resulting in the removal of an entry from memory.	5.0 and later
WSDynamicCacheStats. ExplicitDisk InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations resulting in the removal of an entry from disk.	5.0 and later
WSDynamicCacheStats. LocalExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations generated locally, either programmatically or by a cache policy.	5.0 and later
WSDynamicCacheStats. RemoteExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain.	5.0 and later
WSDynamicCacheStats. RemoteCreationCount	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"	The number of cache entries that are received from cooperating dynamic caches.	5.0 and later

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ObjectsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of cache entries on disk.	6.1
WSDynamicCacheStats. HitsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of requests for cacheable objects that are served from disk.	6.1
WSDynamicCacheStats. ExplicitInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of explicit invalidations resulting in the removal of entries from disk.	6.1
WSDynamicCacheStats. TimeoutInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of disk timeouts.	6.1
WSDynamicCacheStats PendingRemoval FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of pending entries that are to be removed from disk.	6.1
WSDynamicCacheStats. DependencyIdsOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency ID that are on disk.	6.1
WSDynamicCacheStats. DependencyIdsBuffered ForDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency IDs that are buffered for the disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. DependencyIds OffloadedToDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency IDs that are offloaded to disk.	6.1
WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency ID-based invalidations.	6.1
WSDynamicCacheStats. TemplatesOnDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are on disk.	6.1
WSDynamicCacheStats. TemplatesBuffered ForDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are buffered for the disk.	6.1
WSDynamicCacheStats. TemplatesOffloaded ToDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of templates that are offloaded to disk.	6.1
WSDynamicCacheStats. TemplateBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of template-based invalidations.	6.1
WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached.	6.1
WSDynamicCacheStats. OverflowInvalidations FromDisk	WSDynamicCacheStats.NAME - "Servlet: cache_instance_1 " - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit.	6.1

Counter definitions for Object Cache

Name of PMI Statistics	Path	Description	Version
WSDynamicCacheStats. MaxInMemoryCache EntryCount	WSDynamicCacheStats.NAME - "Object: instance_2"	The maximum number of in-memory cache entries.	5.0 and later
WSDynamicCacheStats. InMemoryCache EntryCount	WSDynamicCacheStats.NAME - "Object: instance_2"	The current number of in-memory cache entries	5.0 and later
WSDynamicCacheStats. HitsInMemoryCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats.OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are served from memory.	5.0 and later
WSDynamicCacheStats. HitsOnDiskCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are served from disk.	5.0 and later
WSDynamicCacheStats. ExplicitInvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations.	5.0 and later
WSDynamicCacheStats. LruInvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. instance.	5.0 and later
WSDynamicCacheStats. TimeoutInvalidation Count	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of cache entries that are removed from memory and disk because their timeout has expired.	5.0 and later
WSDynamicCacheStats. InMemoryAndDisk CacheEntryCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The current number of used cache entries in memory and disk.	5.0 and later

Name of PMI Statistics	Path	Description	Version
WSDynamicCacheStats. RemoteHitCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain.	5.0 and later
WSDynamicCacheStats. MissCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that were not found in the cache.	5.0 and later
WSDynamicCacheStats. ClientRequestCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are generated by applications running on this application server.	5.0 and later
WSDynamicCacheStats. DistributedRequest Count	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of requests for cacheable objects that are generated by cooperating caches in this replication domain.	5.0 and later
WSDynamicCacheStats. ExplicitMemory InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations resulting in the removal of an entry from memory.	5.0 and later
WSDynamicCacheStats. ExplicitDisk InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations resulting in the removal of an entry from disk.	5.0 and later
WSDynamicCacheStats. LocalExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations generated locally, either programmatically or by a cache policy.	5.0 and later
WSDynamicCacheStats. RemoteExplicit InvalidationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain.	5.0 and later

Name of PMI Statistics	Path	Description	Version
WSDynamicCacheStats. RemoteCreationCount	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS	The number of cache entries that are received from cooperating dynamic caches.	5.0 and later

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. ObjectsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of cache entries on disk.	6.1
WSDynamicCacheStats. HitsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of requests for cacheable objects that are served from disk.	6.1
WSDynamicCacheStats. ExplicitInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of explicit invalidations resulting in the removal of entries from disk.	6.1
WSDynamicCacheStats. TimeoutInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of disk timeouts.	6.1
WSDynamicCacheStats PendingRemoval FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of pending entries that are to be removed from disk.	6.1
WSDynamicCacheStats. DependencyIdsOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency ID that are on disk.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. DependencyIds BufferedForDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of dependency IDs that are buffered for the disk.	6.1
WSDynamicCacheStats. DependencyIds OffloadedToDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of dependency IDs that are offloaded to disk.	6.1
WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats.DISK_ OFFLOAD_ENABLED	The number of dependency ID-based invalidations.	6.1
WSDynamicCacheStats. TemplatesOnDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are on disk.	6.1
WSDynamicCacheStats. TemplatesBuffered ForDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP / -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The current number of templates that are buffered for the disk.	6.1
WSDynamicCacheStats. TemplatesOffloaded ToDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of templates that are offloaded to disk.	6.1
WSDynamicCacheStats. TemplateBasedInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of template-based invalidations.	6.1
WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached.	6.1

Name of PMI statistics	Path	Description	Version
WSDynamicCacheStats. OverflowInvalidations FromDisk	WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED	The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit.	6.1

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. See Work area partition service for further explanation of this configuration attribute.

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 7. 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 Application Server Performance Monitoring Infrastructure (PMI) helps to monitor and tune the application server performance. Request metrics is another sources of performance data that is provided by WebSphere Application Server. Request metrics allows a request to be timed at WebSphere Application Server component boundaries, enabling a determination of the time that is spent in each major component. For more information about PMI and request metrics, see the *Administering applications and their environment* PDF.

Locating a bottleneck

Consult the following scenarios and suggested solutions:

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

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

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

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

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

Eliminating a bottleneck

Consider the following methods to eliminate a bottleneck:
- Reduce the demand
- Increase resources
- Improve workload distribution
- Reduce synchronization

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

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

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

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

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

threads waiting in the same procedure. Synchronization can often be reduced by: changing the code to only use synchronization when necessary; reducing the path length of the synchronized code; or reducing the frequency of invoking the synchronized code.

Additional references

WebSphere Application Server V6 Scalability and Performance Handbook

WebSphere Application Server Performance Web site

All SPEC jAppServer2004 Results Published by SPEC.

Appendix. Directory conventions

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

These file paths are default locations. You can install the product and other components in any directory where you have write access. You can create profiles in any valid directory where you have write access. Multiple installations of WebSphere Application Server - Express products or components, of course, require multiple locations.

app_server_root - the install_root for WebSphere Application Server
The default installation root directory for WebSphere Application Server - Express is the /QIBM/ProdData/WebSphere/AppServer/V61/Express directory.

profile_root
The default directory for a profile named *profile_name* for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V61/Express/profiles/*profile_name* directory.

app_server_user_data_root - the user_data_root for WebSphere Application Server
The default user data directory for WebSphere Application Server - Express is the /QIBM/UserData/WebSphere/AppServer/V61/Express directory.

plugins_root
The default installation root directory for Web server plug-ins is the /QIBM/ProdData/WebSphere/Plugins/V61/webserver directory.

plugins_user_data_root
The default Web server plug-ins user data root is the /QIBM/UserData/WebSphere/Plugins/V61/webserver directory.

plugins_profile_root
The default Web server plug-ins profile root is the /QIBM/UserData/WebSphere/Plugins/V61/webserver/profiles/*profile_name* directory.

app_client_root
The default installation root directory for the J2EE WebSphere Application Client is the /QIBM/ProdData/WebSphere/AppClient/V61/client directory.

app_client_user_data_root
The default J2EE WebSphere Application Client user data root is the /QIBM/UserData/WebSphere/AppClient/V61/client directory.

app_client_profile_root
The default J2EE WebSphere Application Client profile root is the /QIBM/UserData/WebSphere/AppClient/V61/client/profiles/*profile_name* directory.

web_server_root
The default web server path is /www/*web_server_name*.

shared_product_library
The shared product library, which contains all of the objects shared by all Version 6.1 installations on the system, is QWAS61. This library contains objects such as the product definition, the subsystem description, the job description, and the job queue.

product_library
The product library, which contains program and service program objects (similar to .exe, .dll, .so objects for Windows, Linux, and UNIX operating system platforms), is: QWAS61*x* where *x* is A, B, C, ... For the default installation, the value is QWAS61A.

product_lib

The product library that contains the service program objects for the web server plugins. For the default Web Server Plugins install, this is QWAS61A. If you install the Web Server Plugins multiple times, the product_lib is QWAS61c, where *c* is B, C, D, ... The plugins_install_root/properties/product.properties contains the value for the product library..

cip_app_server_root

The default installation root directory is the /QIBM/ProdData/WebSphere/AppServer/V61/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

A CIP is a WebSphere Application Server - Express product bundled with optional maintenance packages, an optional configuration archive, one or more optional enterprise archive files, and other optional files and scripts.

cip_user_data_root

The default user data root directory is the /QIBM/UserData/WebSphere/AppServer/V61/Express/cip/*cip_uid* directory for a customized installation package (CIP) produced by the Installation Factory.

cip_profile_root

The default profile root directory is the /QIBM/UserData/WebSphere/AppServer/V61/Express/cip/*cip_uid*/profiles/*profile_name* directory for a customized installation package (CIP) produced by the Installation Factory.

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

Trademarks and service marks

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