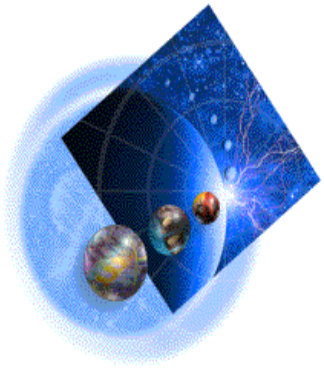
IBM WebSphere Application Server, Advanced Edition

Tuning Guide



What's new for performance tuning?

Performance tuner wizard

Dynamic fragment caching

Symptom table

Tuning basics

What influences tuning?

Types of tuning

Parameter tuning

Tuning parameters with a high impact

Tuning parameters for avoiding failures

Adjusting WebSphere Application Server system queues

WebSphere queuing network

Closed queues versus open queues

Queue settings in WebSphere

Determining settings

Queuing before WebSphere

Drawing a throughput curve

Queue adjustments

Queue adjustments for accessing patterns

Queuing and enterprise beans

Queuing and clustering

Tuning Secure Socket Layer

Overview of handshake and bulk encryption/decryption

How to enhance Secure Socket Layer performance

Application assembly performance checklist

Tuning Java memory

The garbage collection bottleneck

The garbage collection gauge

Detecting over utilization of objects

Detecting memory leaks

Java heap parameters

Number of connections to DB2

Solaris TCP parameters

Workload management topology

Individual performance parameters

Hardware capacity and settings

Processor speed

Memory

Network

Operating system settings

Windows NT/2000 TCP/IP parameters

Windows NT/2000 TcpTimedWaitDelay

Windows NT/2000 MaxUserPort

AIX (4.3.3)

AIX file descriptors (ulimit)

Solaris

Solaris file descriptors (ulimit)

Solaris tcp_time_wait_interval

Solaris tcp_fin_wait_2_flush_interval

Solaris tcp_keepalive_interval

Other Solaris TCP parameters

Solaris kernel semsys:seminfo_semume

Solaris kernel semsys:seminfo_semopm

HP-UX 11

Setting the Virtual Page Size to 64K for WebSphere Application Server JVM

HP-UX 11 tcp_conn_request_max

HP-UX 11 kernel parameter recommendations

The Web server

Web server configuration reload interval

IBM HTTP server - AIX and Solaris

MaxClients

MinSpareServers, MaxSpareServers, and StartServers

Netscape Enterprise server - AIX and Solaris

Active threads

Microsoft Internet Information Server - Windows NT/2000

Internet Information Server (IIS) Permission Properties

Number of expected hits per day

ListenBackLog parameter

IBM HTTP server - Linux

MaxRequestsPerChild

IBM HTTP server - Windows NT/2000

ThreadsPerChild

ListenBackLog

The WebSphere application server process

Adjust the application server process priority

Web containers

Web container maximum thread size

Transport Maximum Keep-Alive

Transport Maximum Requests per Keep-Alive

URL invocation cache

Allow thread allocation beyond maximum

EJB container

Cache settings

Break CMP enterprise beans into several enterprise bean modules

Security

Turn off security when you do not need it

Fine-tune the security cache time out for the environment

Security cache types and sizes (system parameters)

Configure Secure Socket Layer sessions appropriately

Object Request Broker (ORB)

Pass-by-value versus pass-by-reference (NoLocalCopies)

com.ibm.CORBA.ServerSocketQueueDepth

com.ibm.CORBA.MaxOpenConnections and ORB Connection Cache Maximum

Object Request Broker thread pool size

Using JNI ReaderManager and Reader Threads

Java Virtual Machines (JVMs)

Sun JDK 1.3 HotSpot -server warmup

Sun JDK 1.3 HotSpot new generation pool size

Just In Time (JIT) compiler

Heap size settings

Class garbage collection

The database

Database location

WebSphere data source connection pool size

Prepared statement cache size

DB2

Use TCP sockets for DB2 on Linux

DB2 MaxAppls

DB2 MaxAgents

DB2 buffpage

DB2 query optimization level

DB2 reorgchk

DB2 MinCommit

Session management

WebSphere Application Server Enterprise Extensions Message Listener

Maximum sessions

Multiple application servers listening on the same queue

Additional reference

Performance tool procedures

Starting Windows NT/2000 Performance Monitor

What's new for performance tuning?

Performance tuner wizard
Dynamic fragment caching

Performance tuner wizard

The Performance tuner wizard is a tool included in WebSphere Application Server, Advanced Edition that includes the most common performance related settings associated with the application server. Use Performance Tuner Wizard to optimize the settings for applications, servlets, enterprise beans, data sources and expected load. Parameters that can be set include:

- Web container - maximum thread size
- ORB properties - pass-by-reference, ORB thread pool size
- Data source - connection pool size, statement cache size
- Database (DB2 only) - This calls the DB2SmartGuide, which, in turn, tunes the DB2 database associated with the data source
- Java Virtual Machine (JVM) - initial and maximum heap size

To invoke this from the administrative console select **Console** > **Wizards** > **Performance Tuner**.

Dynamic fragment caching

Dynamic fragment caching is the ability to cache the output of dynamic servlets and JSP files, a technology that dramatically improves application performance. This cache, working within the JVM of an application server, intercepts calls to a servlet service() method, checking whether it can serve the invocation from the cache rather than re-execute the servlet. Because J2EE applications have high read-write ratios and can tolerate a small degree of latency in the freshness of their data, fragment caching can create an opportunity for dramatic gains in server response time, throughput and scalability.

Once a servlet has been executed (generating the output that will be cached), a cache entry is generated containing that output. Also generated are side effects of the execution (that is, invoking other servlets or JSP files), as well as metadata about the entry, including time out and entry priority information. Unique entries are distinguished by an ID string generated from the HttpServletRequest object for each invocation of the servlet. This results in a servlet being cached depending on request parameters that the URI used to invoke the servlet or session information. Because JSP files are compiled by WebSphere Application Server into servlets, JSPs and servlets are used interchangeably (except when declaring elements within an XML file).

To set this:

1. From the administrative console select the application server you are tuning.
2. Click **Services** > **Web Container Service** > **Edit Properties**.
3. Select the **Servlet Caching** tab and check the box **Enable Servlet Caching**. By checking this box, you are enabling the caching facility, however, nothing will be cached until specific servlets/JSP files are selected.
4. Click **OK** and save the changes.
5. Restart the application server.

Symptom table

Take a shortcut into tuning by reviewing the symptom table. The table is designed for easy access to symptoms and a quick link to tuning information related to that symptom. The table contains the following types of information:

- **The symptom**
  The left column lists symptoms and descriptions. The symptoms can be specific or general.
- **Additional information**
  The right column lists a link that leads to additional information about the symptom.

Symptom	Additional information
Throughput and response time are undesirable.	Processor speed
AIX: Memory allocation error Solaris: Too many files open	AIX file descriptors (ulimit) or Solaris file descriptors (ulimit)
Solaris: The server stalls during peak periods, responses take minutes, processor utilization remains high with all activity in the system processes, and netstat shows many sockets are open to port 80 in CLOSE_WAIT or FIN_WAIT_2 state.	Solaris tcp_time_wait_interval and Solaris tcp_fin_wait_2_flush_interval
Windows NT/2000: Netstat shows too many sockets are in TIME_WAIT.	Windows NT/2000 TcpTimedWaitDelay
Throughput is undesirable and the application server priority has not been adjusted.	Adjusting the operating system priority of the WebSphere Application Server process
Under load, client requests do not arrive at the Web server because they time out or are rejected.	For HP-UX 11 see HP-UX 11 tcp_conn_request_max For IIS Windows NT/2000 see ListenBackLog parameter For IBM HTTP Server on NT see ListenBackLog
Windows NT/2000: WebSphere Application Server performance decreased after an application server from another vendor was installed.	IIS Permission Properties

Resource Analyzer percent maxed metric indicates that the Web container thread pool is too small.	Web container maximum thread size
Netstat shows too many TIME_WAIT state sockets for port 9080.	Web container transport maximum Keep-Alive
Too much disk input/output occurs due to paging.	Heap size settings
Resource Analyzer's percent used metric for a data source connection pool indicates the pool size to too large.	WebSphere data source connection pool size
Resource Analyzer's prepared statement cache discards metric indicates the data source prepared statement cache size is too small.	Prepared statement cache size
Too much disk input/output occurs due to DB2 writing log records.	DB2 MinCommit
Resource Analyzer percent maxed metric indicates the Object Request Broker thread pool is too small.	Queuing and enterprise beans
Resource Analyzer Java Virtual Machine Profiler Interface (JVMPI) indicates over-utilization of objects when too much time is being spent in garbage collection.	Detecting over-utilization of objects
Resource Analyzer used memory metric shows memory leaks and Java displays an `Out of Memory` exception.	Detecting memory leaks
Throughput, response time and scalability are undesirable.	If the application permits, exploit dynamic fragment caching

Tuning basics

A wide range of performance improvements can be made to WebSphere Application Server when using procedures available for tuning. This tuning guide teaches how tuning works, by giving general recommendations and a description of specific tuning methodologies. Also available are hints and tips on the various factors and variables that can enhance performance tuning.

Use the tuning guide, along with its examples and resources, to expand your tuning experience. Tuning is an ongoing learning process. Results can vary from those presented in this guide.

What influences tuning?

The following are all components that can affect the performance of WebSphere Application Sever:

- The application being used
- Hardware capacity and settings
- Operating system settings
- Web server
- WebSphere application server process
- Java Virtual Machine (JVM)
- Database

Each has its own tuning options, varying in importance and impact. Each of the above are explained in detail in the Individual Performance Parameters section of this document.

For convenience, some procedures are described for setting parameters in other products. Because these products can change, consider these procedures as suggestions.

Types of tuning

There are two types of tuning: application tuning and parameter tuning.

Although application tuning sometimes offers the greatest tuning improvements, this document focuses on individual performance parameters and synergy between them.

The white paper, WebSphere Application Server Development Best Practices for Performance and Scalability, addresses application tuning by describing development best practices for both Web applications containing servlets, Java Server Pages (JSP) files, Java Database Connectivity (JDBC), and enterprise applications containing enterprise bean components.

Parameter tuning

Parameter tuning is the art of changing WebSphere Application Server settings with the goal of improving performance. The values suggested in this document are general guidelines. The optimal settings for your environment can vary significantly. In addition, remember that after tuning one bottleneck away, you can encounter another, unrelated bottleneck. If so, you might not experience the performance improvement until both bottlenecks have been removed.

This section discusses two kinds of tuning parameters:

- Tuning parameters with a high impact
- Tuning parameters for avoiding failure

Tuning parameters with a high impact

These parameters have high performance results. They are a subset of all other parameters and have an important impact on performance. Because these are application dependent, the appropriate parameter settings for the application and environment might be different.

The following table lists various high performance-enhancing tuning parameters:

Application assembly performance checklist
Adjusting WebSphere Application Server system queues
Using pass-by-value versus pass-by-reference (NoLocalCopies)
Adjusting Solaris TCP parameters
Tuning Java memory
Adjusting MaxRequestsPerChild: on Linux with IBM HTTP Server
Adjusting WebSphere data source connection pool size
Adjusting prepared statement cache size
Web server configuration reload interval

Tuning parameters for avoiding failures

The following parameters help to prevent functional problems:

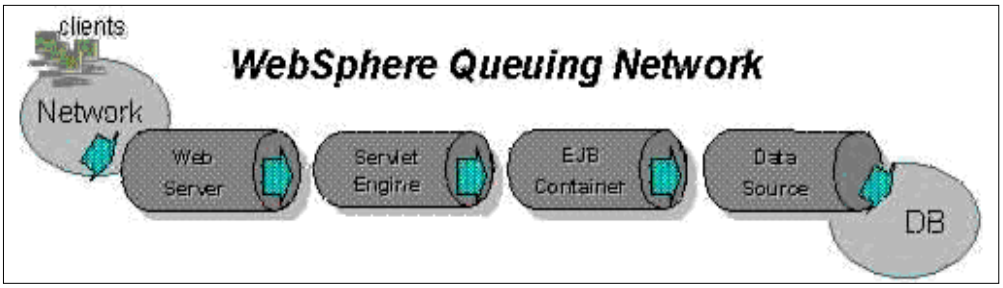
ListenBackLog parameter: Applies if running Windows NT/2000 with IIS under heavy client load
Number of connections to DB2: If you establish more connections than DB2 sets up by default
Allow thread allocation beyond maximum has been selected and the system is overloaded because too many threads are allocated.
Using TCP Sockets for DB2 on Linux: For local databases
WebSphere data source connection pool size: Be sure to have enough connections to handle the extra connections required for transaction processing with Entity EJBs and to avoid deadlock.

Adjusting the queues in WebSphere Application Server

WebSphere Application Server has a series of interrelated components that must be harmoniously tuned to support the custom needs of your end-to-end e-business application. These adjustments help the system achieve maximum throughput while maintaining the overall stability of the system.

Queuing network

WebSphere Application Server establishes a queuing network, which is a network of interconnected queues that represent the various components of the application serving platform. These queues include the network, Web server, Web container, EJB container, data source and possibly a connection manager to a custom back-end system. Each of these resources represents a queue of requests waiting to use that resource.



The WebSphere queues are load-dependent resources. The average service time of a request depends on the number of concurrent clients.

Closed queues versus open queues

Most of the queues that make up the queuing network are closed queues. In contrast with an open queue, a closed queue places a limit on the maximum number of requests present in the queue.

A closed queue allows system resources to be tightly managed. For example, the Web container's Max Connections setting controls the size of the Web container queue. If the average servlet running in a Web container creates 10MB of objects during each request, then a value of 100 for max connections would limit the memory consumed by the Web container to 1GB.

In a closed queue, a request can be in one of two states: active or waiting. An active request is either doing work or waiting for a response from a downstream queue. For example, an active request in the Web server is either doing work (such as retrieving static HTML) or waiting for a request to complete in the Web container. In the waiting state, the request is waiting to become active. The request remains in the waiting state until one of the active requests leaves the queue.

All Web servers supported by WebSphere Application Server are closed queues, as are WebSphere Application Server data sources. Web containers can be configured as either open or closed queues. In general, it is best to make them closed queues. EJB containers are open queues; if there are no threads available in the pool, a new one will be created for the duration of the request.

If enterprise beans are being called by servlets, the Web container limits the number of total concurrent requests into an EJB container because the Web container also has a limit. This is true only if enterprise beans are called from the servlet thread of execution. Nothing prevents you from creating threads and bombarding the EJB container with requests. This is one reason why it is not a good idea for a servlet to create its own work threads.

Queue settings in WebSphere Application Server

The following outlines the various queue settings:

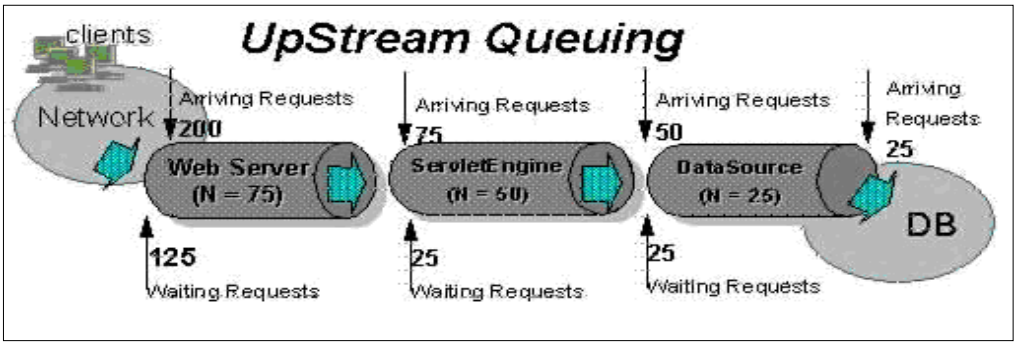
- IBM HTTP Server: MaxClients (for UNIX) and ThreadsPerChild (for Windows NT/2000)
  Microsoft Internet Information Server: MaxPoolThreads and PoolThreadLimit
- Web container
  - Max Connections
  - Transport Maximum Keep-Alive
  - Transport Maximum Requests per Keep-Alive
- Object Request Broker thread pool size
- Data source connection pool size
- Prepared statement cache size

Determining the settings

The following section outlines a methodology for configuring the WebSphere Application Server queues. The dynamics of the system can be changed and therefore, the tuning parameters can be changed, by moving resources (for example, moving the database server onto another machine) or providing more powerful resources, such as a faster set of CPUs with more memory. Thus, adjust the tuning parameters to a specific configuration of the production environment.

Queuing before WebSphere

The first rule of tuning is to minimize the number of requests in WebSphere Application Server queues. In general, it is better for requests to wait in the network (in front of the Web server) than it is for them to wait in WebSphere Application Server. This configuration will result in allowing into the queuing network only requests that are ready to be processed. To accomplish this configuration, specify the queues furthest upstream (closest to the client) to be slightly larger, and specify the queues further downstream (furthest from the client) to be progressively smaller.



The queues in this queuing network are progressively smaller as work flows downstream. When 200 clients arrive at the Web server, 125 requests remain queued in the network because the Web server is set to handle 75 concurrent clients. As the 75 requests pass from the Web server to the Web container, 25 remain queued in the Web server and the remaining 50 are be handled by the Web container. This process progresses through the data source until 25 users arrive at the final destination, the database server. Because, at each point upstream, there is some work waiting to enter a component, no component in this system must wait for work to arrive. The bulk of the requests wait in the network, outside of WebSphere Application Server. This adds stability because no component is overloaded. Routing software like IBM's Network Dispatcher can be used to direct waiting users to other servers in a WebSphere Application Server cluster.

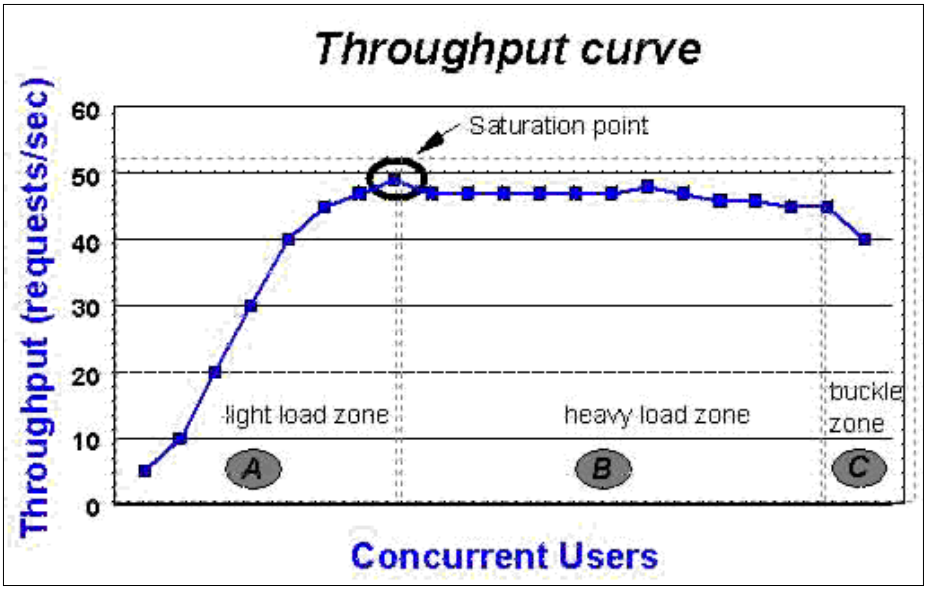
Drawing a throughput curve

Using a test case that represents the full spirit of the production application (for example, it exercises all meaningful code paths) or using the production application itself, run a set of experiments to determine when the system capabilities are maximized (the **saturation point**). Conduct these tests after most of the bottlenecks have been removed from the application. The typical goal of these tests is to drive CPUs to near 100% utilization.

Start the initial baseline experiment with large queues. This allows maximum concurrency through the system. For example, start the first experiment with a queue size of 100 at each of the servers in the queuing network: Web server, Web container and data source.

Next, begin a series of experiments to plot a throughput curve, increasing the concurrent user load after each experiment. For example, perform experiments with 1 user, 2 users, 5, 10, 25, 50, 100, 150 and 200 users. After each run, record the throughput (requests per second) and response times (seconds per request).

The curve resulting from the baseline experiments should resemble the typical throughput curve shown as follows:



The throughput of WebSphere Application Server is a function of the number of concurrent requests present in the total system. Section A, the light load zone, shows as the number of concurrent user requests increases, the throughput increases almost linearly with the number of requests. This reflects that, at light loads, concurrent requests face very little congestion within the WebSphere Application Server system queues. At some point, congestion starts to develop and throughput increases at a much lower rate until it reaches a saturation point that represents the maximum throughput value, as determined by some bottleneck in the WebSphere Application Server system. The most manageable type of bottleneck is when the CPUs of the WebSphere Application Server machines become saturated. This is desirable because a CPU bottleneck is remedied by adding additional or more powerful CPUs.

In the heavy load zone or Section B, as the concurrent client load increases, throughput remains relatively constant. However, the response time increases proportionally to the user load. That is, if the user load is doubled in the heavy load zone, the response time doubles. At some point, represented by Section C, the buckle zone, one of the system components becomes exhausted. At this point, throughput starts to degrade. For example, the system might enter the buckle zone when the network connections at the Web server exhaust the limits of the network adapter or if the operating system limits for file handles is exceeded.

If the saturation point is reached by driving the use of the system CPUs close to 100%, move on to the next step. If the CPU is not driven to 100%, there is likely a bottleneck that is being aggravated by the application. For example, the application might be creating Java objects excessively causing garbage collection bottlenecks in Java.

There are two ways to managing application bottlenecks: remove the bottleneck or clone the bottleneck. The best way to manage a bottleneck is to remove it. Use a Java-based application profiler to examine the overall object utilization. Profilers such as Performance Trace Data Visualizer(PTDV), JProbe and Jinsight can be used.

Queue adjustments

The number of concurrent users at the throughput saturation point represents the maximum concurrency of the application. For example, if the application saturated WebSphere Application Server at 50 users, you might find that 48 users gave the best combination of throughput and response time. This value is called the Max Application Concurrency value. Max Application Concurrency becomes the value to use as the basis for adjusting the WebSphere Application Server system queues. Remember, it is desirable for most users to wait in the network, therefore, decrease queue sizes while moving downstream farther from the client. For example, given a Max Application Concurrency value of 48, start with system queues at the following values: Web server 75, Web container 50, data source 45. Perform a set of additional experiments adjusting these values slightly higher and lower to find the best settings.

The Resource Analyzer can be used to determine the number of concurrent users through the servlet engine thread pool concurrently active threads metric.

In performance experiments, throughput has increased by 10-15% when the Web container transport maximum Keep-Alive are adjusted to match the maximum number of Web container threads.

Adjusting queue settings for access patterns

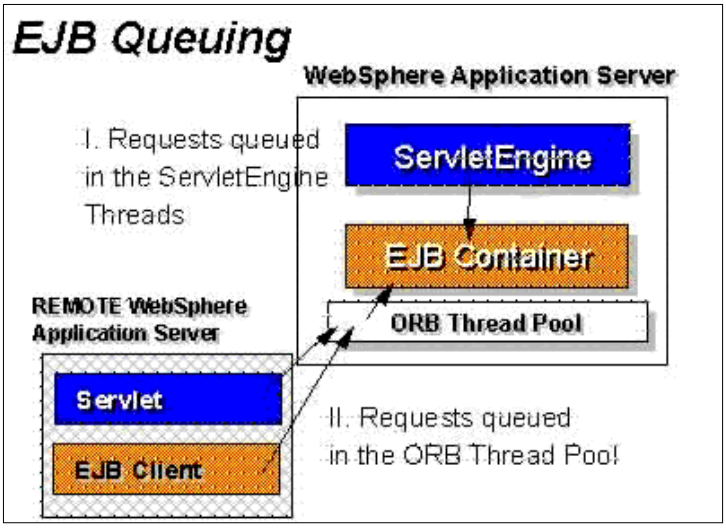
In many cases, only a fraction of the requests passing through one queue enters the next queue downstream. In a site with many static pages, many requests are fulfilled at the Web server and are not passed to the Web container. In this circumstance, the Web server queue can be significantly larger than the Web container queue. In the previous section, the Web server queue was set to 75 rather than closer to the value of Max Application Concurrency. Similar adjustments need to be made when different components have different execution times.

For example, in an application that spends 90% of its time in a complex servlet and only 10% making a short JDBC query, on average 10% of the servlets are using database connections at any time, so the database connection queue can be significantly smaller than the Web container queue. Conversely, if much of a servlet execution time is spent making a complex query to a database, consider increasing the queue values at both the Web container and the data source. Always monitor the CPU and memory utilization for both the WebSphere Application Server and the database servers to ensure the CPU or memory are not being saturated.

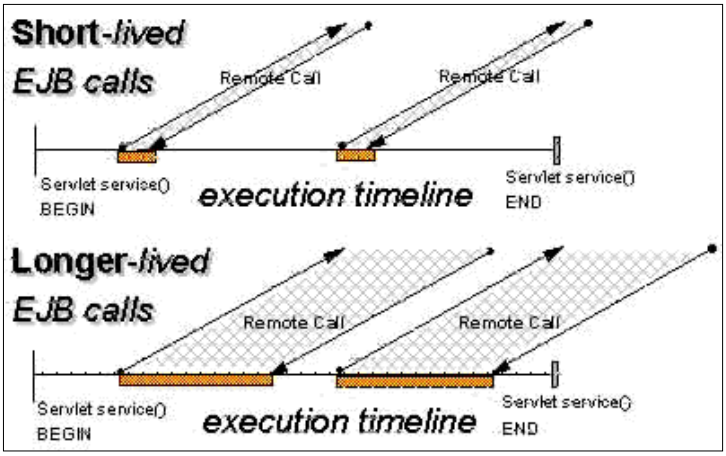
Queuing and enterprise beans

Method invocations to enterprise beans are queued only if the client, which is making the method call, is remote. For example, if the EJB client is running in a separate Java Virtual Machine (another address space) from the enterprise bean. In contrast, if the EJB client (either a servlet or another enterprise bean) is installed in the same JVM, the EJB method runs on the same thread of execution as the EJB client and there is no queuing.

Remote enterprise beans communicate by using the RMI/IIOP protocol. Method invocations initiated over RMI/IIOP are processed by a server-side ORB. The thread pool acts as a queue for incoming requests. However, if a remote method request is issued and there are no more available threads in the thread pool, a new thread is created. After the method request completes the thread is destroyed. Therefore, when the ORB is used to process remote method requests, the EJB container is an open queue, because its use of threads is unbounded. The following illustration depicts the two queuing options of enterprise beans.



When configuring the thread pool, it is important to understand the calling patterns of the EJB client. If a servlet is making a small number of calls to remote enterprise beans and each method call is relatively quick, consider setting the number of threads in the ORB thread pool to a value lower than the Web container thread pool size value.

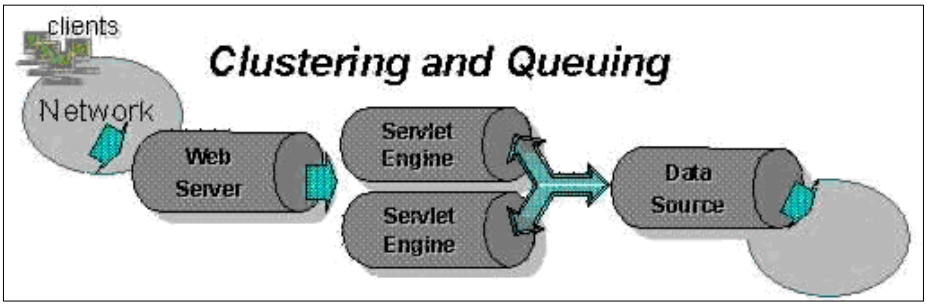


Resource Analyzer shows a metric called percent maxed used to determine how much of the time all of the configured threads are in use. If this value is consistently in the double-digits, then the ORB could be a bottleneck and the number of threads should be increased.

The degree to which the ORB thread pool value needs to be increased, is a function of the number of simultaneous servlets (that is, clients) calling enterprise beans and the duration of each method call. If the method calls are longer, consider making the ORB thread pool size equal to the Web container size because there is little interleaving of remote method calls. If the servlet makes only short-lived or quick calls to the ORB, the ORB thread pool can be small. Several servlets can potentially reuse the same ORB thread. In this case, the ORB thread pool can be small, perhaps even one-half of the thread pool size setting of the Web container. If the application spends a lot of time in the ORB, configure a more even relationship between the Web container and the ORB.

Queuing and clustering

The capabilities for cloning application servers can be a valuable asset in configuring highly scalable production environments. This is especially true when the application is experiencing bottlenecks that are preventing full CPU utilization of Symmetric Multiprocessing (SMP) servers. When adjusting the WebSphere Application Server system queues in clustered configurations, remember that when a server is added to a cluster, the server downstream receives twice the load.



Two Web container clones are located between a Web server and a data source. It is assumed the Web server, servlet engines and data source (but not the database) are all running on a single SMP server. Given these constraints, the following queue considerations need to be made:

- Web server queue settings can be doubled to ensure ample work is distributed to each Web container.
- Web container thread pools can be reduced to avoid saturating a system resource such as CPU or another resource that the servlets are using.
- The data source can be reduced to avoid saturating the database server.
- Java heap parameters can be reduced for each instance of the application server. For versions of the JVM shipped with WebSphere Application Server, it is crucial that the heap from all JVMs remain in physical memory. Therefore, if a cluster of four JVMs are running on a system, enough physical memory must be available for all four heaps.

Tuning Secure Socket Layer

The following are two types of Secure Socket Layer(SSL) performance:

- Handshake
- Bulk encryption/decryption

Overview of handshake and bulk encryption/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.

How to enhance SSL performance

In order to enhance SSL performance, the number of individual SSL connections and handshakes must be decreased.

Decreasing the number of connections increases performance for secure communication through SSL connections, as well as non-secure communication through simple TCP connections. One way to decrease individual SSL connections is to use a browser that supports HTTP 1.1. Decreasing individual SSL connections could be impossible for some users if they cannot upgrade to HTTP 1.1.

It is more common to decrease the number of connections (both TCP and SSL) between two WebSphere Application Server components. The following guidelines help to ensure 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:

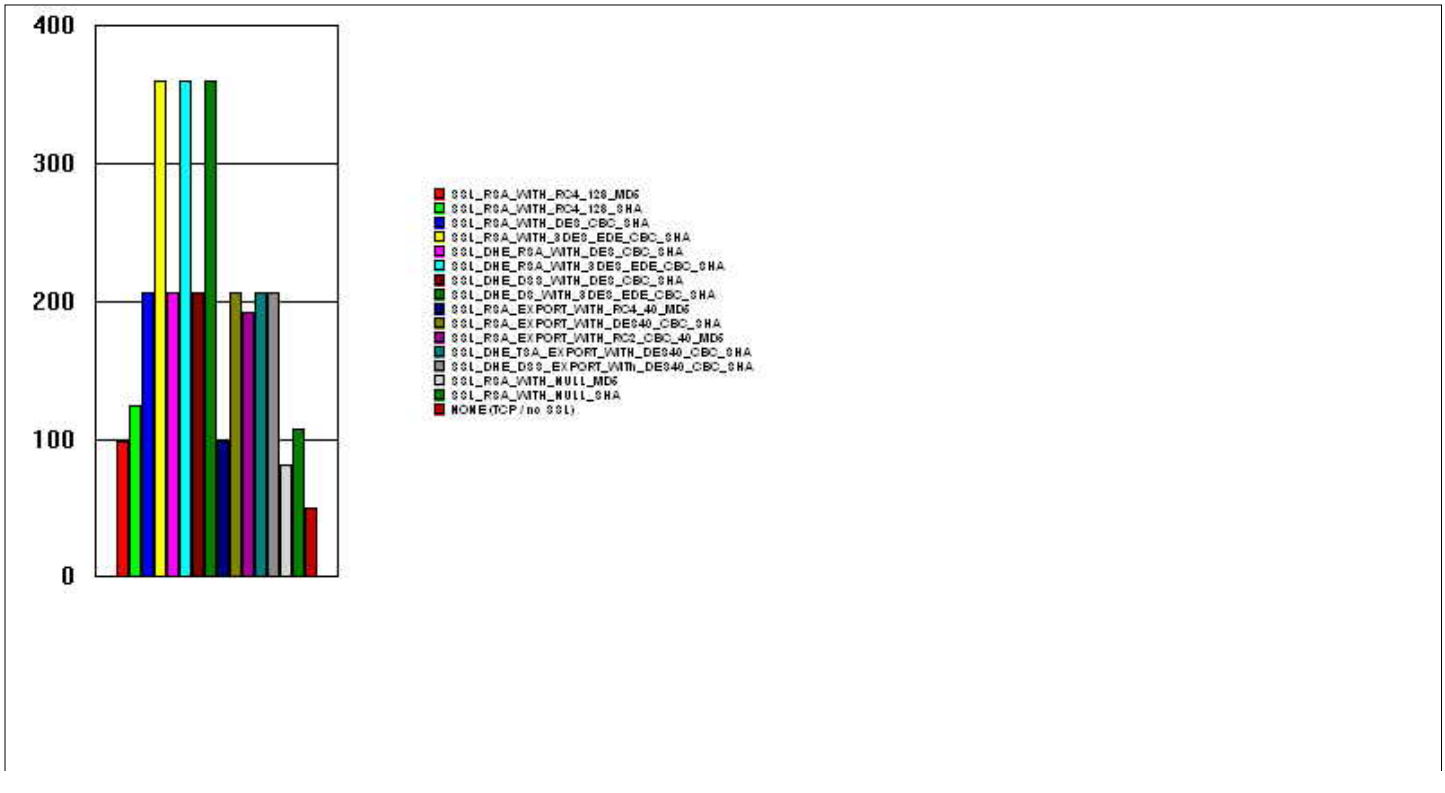
- The maximum number of Keep-Alives should be, at minimum, as large as the maximum number of requests per thread of the Web server (or maximum number of processes for IHS on UNIX). In other words, make sure the Web server plug-in is capable of obtaining a Keep-Alive connection for every possible concurrent connection to the application server. Otherwise, the application server will close the connection after a single request has been processed. Also, the maximum number of threads in the Web container thread pool should be larger than the maximum number of Keep-Alives, in order to prevent the Web container threads from being consumed with Keep-Alive connections.
- The maximum number of requests per Keep-Alive connection can also be increased. The default value is 100, which means the application server will close the connection from the plug-in after 100 requests. The plug-in would then have to open a new connection. The purpose of this parameter is to prevent denial of service attacks when connecting to the application server and continuously send requests in order 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/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; these connections do not benefit from a hardware device which only accelerates SSL handshakes.
- 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 it will perform better with hardware. Some algorithms are typically inefficient in hardware (for example, DES and 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 test software calculating the data used IBM JSSE for both the client and server software, and used no crypto hardware support. The test did not include the time to establish a connection, but 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 enabled a single cipher suite for each test case. After the connection was established, the client timed how long it took 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. The following chart shows the performance of each cipher suite.



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 SHA are the two hash algorithms used to provide data integrity. MD5 is 25% 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.

Application assembly performance checklist

Application assembly tools in WebSphere Application Server, Version 4.0 are used to assemble J2EE components and modules into J2EE applications. Generally, this consists of defining application components and their attributes including enterprise beans, servlets and resource references. Many of these application configuration settings and attributes play an important role with respect to the runtime performance of the deployed application. Below are a checklist of settings, along with a "guide" to help configure optimal settings.

The checklist includes these settings:

- Enterprise bean modules
  - Entity EJBs - Bean cache

        - Method extensions - Isolation level
        - Method extensions - Access intent
        - Container transactions
    - Web module
        - Web application - Distributable
        - Web application - Reload interval
        - Web application - Reload enabled
        - Web application - Web components - Load on startup

Enterprise bean modules

Entity EJBs - Bean cache

WebSphere Application Server provides significant flexibility in the management of database data with Entity EJBs. The Entity EJBs **Activate At** and **Load At** configuration settings specify how and when to load and cache data from the corresponding database row data of an enterprise bean. These configuration settings provide the capability to specify enterprise bean commit options A, B or C, as specified in the EJB 1.1 specification.

Guide

The **Activate At** and **Load At** settings are detailed below along with specific settings to achieve each of the enterprise bean commit options A, B and C.

Commit option A provides maximum enterprise bean performance by caching database data outside of the transaction scope. Generally, commit option A is only applicable where the EJB container has exclusive access to the given database. Otherwise, data integrity is compromised. Commit option B provides more aggressive caching of Entity EJB object instances, which can result in improved performance over commit option C, but also results in greater memory usage. Commit option C is the most common real-world configuration for Entity EJBs.

- **Bean cache - Activate At** This setting specifies the point at which an enterprise bean is activated and placed in the cache. Removal from the cache and passivation is also governed by this setting. Valid values are **Once** and **Transaction**. **Once** indicates that the bean is activated when it is first accessed in the server process, and passivated (and removed from the cache) at the discretion of the container, for example, when the cache becomes full. **Transaction** indicates that the bean is activated at the start of a transaction and passivated (and removed from the cache) at the end of the transaction. The default value is **Transaction**.
- **Bean cache - Load At** This setting specifies when the bean loads its state from the database. The value of this property implies whether the container has exclusive or shared access to the database. Valid values are **Activation** and **Transaction**. **Activation** indicates the bean is loaded when it is activated (**Once** or **Transaction**) and implies that the container has exclusive access to the database. **Transaction** indicates that the bean is loaded at the start of a transaction and implies that the container has shared access to the database. The default is **Transaction**.

  The settings of the **Activate At** and **Load At** properties govern which commit options are used.
    - For commit option A (implies exclusive database access), use **Activate At = Once** and **Load At = Activation**. This option reduces database input/output (avoids calls to the ejbLoad function) but serializes all transactions accessing the bean instance. Option A can increase memory usage by maintaining more objects in the cache, but can provide better response time if bean instances are not generally accessed concurrently by multiple transactions.
    - For commit option B (implies shared database access), use **Activate At = Once** and **Load At = Transaction**. Option B can increase memory usage by maintaining more objects in the cache. However, because each transaction creates its own copy of an object, there can be multiple copies of an instance in memory at any given time (one per transaction), requiring the database be accessed at each transaction. If an enterprise bean contains a significant number of calls to the ejbActivate function, using option B can be beneficial because the required object is already in the cache. Otherwise, this option does not provide significant benefit over option A.
    - For commit option C (implies shared database access), use **Activate At = Transaction** and **Load At = Transaction**. This option can reduce memory usage by maintaining fewer objects in the cache, however, there can be multiple copies of an instance in memory at any given time (one per transaction). This option can reduce transaction contention for enterprise bean instances that are accessed concurrently but not updated.

Method extensions - Isolation level

WebSphere Application Server enterprise bean method extensions provide settings to specify the level of transactional isolation used when accessing data. The valid values are:

- Serializable
- Repeatable Read
- Read Committed
- Read Uncommitted

Guide

Defined below, isolation level settings specify various degrees of runtime data integrity provided by the corresponding database. First, choose a setting that meets data integrity requirements for the given application and specific database characteristics.

Isolation level also plays an important role in performance. Higher isolation levels reduce performance by increasing row locking and database overhead while reducing data access concurrency. Various databases provide different behavior with respect to the isolation settings. In general, **Repeatable Read** is an appropriate setting for DB2 databases. **Read Committed** is generally used for Oracle. Oracle does not support **Repeatable Read** and will translate this setting to the highest isolation level serializable.

Isolation level can be specified at the bean or method level. Therefore, it is possible to configure different isolation settings for various methods. This is an advantage when some methods require higher isolation than others, and can be used to achieve maximum performance while maintaining integrity requirements. However, isolation cannot change between method calls within a single enterprise bean transaction. A runtime exception will be thrown in this case.

Isolation levels

Serializable
This level prohibits the following types of reads:

- **Dirty reads:** A transaction reads a database row containing uncommitted changes from a second transaction.
- **Nonrepeatable reads:** One transaction reads a row, a second transaction changes the same row, and the first transaction rereads the row and gets a different value.
- **Phantom reads:** One transaction reads all rows that satisfy an SQL WHERE condition, a second transaction inserts a row that also satisfies the WHERE condition, and the first transaction applies the same WHERE condition and gets the row inserted by the second transaction.

Repeatable Read
This level prohibits dirty reads and nonrepeatable reads, but it allows phantom reads.

Read Committed
This level prohibits dirty reads, but allows nonrepeatable reads and phantom reads.

Read Uncommitted
This level allows dirty reads, nonrepeatable reads, and phantom reads.

The container uses the transaction isolation level attribute as follows:

- Session beans and entity beans with bean-managed persistence (BMP).
  For each database connection used by the bean, the container sets the transaction isolation level at the start of each transaction unless the bean explicitly sets the isolation level on the connection.
- Entity beans with container-managed persistence (CMP).
  The container generates database access code that implements the specified isolation level.

Method extensions - Access intent

WebSphere Application Server enterprise bean method extensions provide settings to specify individual enterprise bean methods as read-only. This setting is used to denote whether the method can update entity attribute data (or invoke other methods that can update data in the same transaction).

Guide

By default, all enterprise bean methods are assumed to be "update" methods. This results in EJB Entity data always being persisted back to the database at the close of the enterprise bean transaction. Marking enterprise methods as Access Intent Read that do not update entity attributes, provides a significant performance improvement by allowing the WebSphere Application Server EJB container to skip the (unnecessary) database update. **Note:** WebSphere Application Server, Version 4.01 provides an added behavior for "finder" methods for CMP Entity EJBs. By default, WebSphere Application Server will invoke a "Select for Update" query for CMP enterprise bean finder methods such as findByPrimaryKey. This exclusively locks the database row for the duration of the enterprise bean transaction. However, if the enterprise bean finder method has been marked as Access Intent Read, the container will not issue the "For Update" on the select resulting in only a read lock on the database row.

Container transactions

The container transaction properties specifies how the container manages transaction scopes when delegating invocation to the enterprise bean individual business method. The legal values are:

- Never
- Mandatory
- Requires New
- Required
- Supports
- Not Supported
- Bean Managed

Guide

Container transaction attribute can be specified individually for one or more enterprise bean methods. Enterprise bean methods not requiring transactional behavior can be configured as **Supports** to reduce container transaction management overhead.

Legal values

Never
This legal value directs the container to invoke bean methods without a transaction context. If the client invokes a bean method from within a transaction context, the container throws the `java.rmi.RemoteException` exception.

If the client invokes a bean method from outside a transaction context, the container behaves in the same way as if the **Not Supported** transaction attribute was set. The client must call the method without a transaction context.

Mandatory
This legal value directs the container to always invoke the bean method within the transaction context associated with the client. If the client attempts to invoke the bean method without a transaction context, the container throws the `javax.jts.TransactionRequiredException` exception to the client. The transaction context is passed to any enterprise bean object or resource accessed by an enterprise bean method.

Enterprise bean clients that access these entity beans must do so within an existing transaction. For other enterprise beans, the enterprise bean or bean method must implement the **Bean Managed** value or use the **Required** or **Requires New** value. For non-enterprise bean EJB clients, the client must invoke a transaction by using the javax.transaction.UserTransaction interface.

Requires New
This legal value directs the container to always invoke the bean method within a new transaction context, regardless of whether the client invokes the method within or outside a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Required
This legal value directs the container to invoke the bean method within a transaction context. If a client invokes a bean method from within a transaction context, the container invokes the bean method within the client transaction context. If a client invokes a bean method outside a transaction context, the container creates a new transaction context and invokes the bean method from within that context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Supports
This legal value directs the container to invoke the bean method within a transaction context if the client invokes the bean method within a transaction. If the client invokes the bean method without a transaction context, the container invokes the bean method without a transaction context. The transaction context is passed to any enterprise bean objects or resources that are used by this bean method.

Not Supported
This legal value directs the container to invoke bean methods without a transaction context. If a client invokes a bean method from within a transaction context, the container suspends the association between the transaction and the current thread before invoking the method on the enterprise bean instance. The container then resumes the suspended association when the method invocation returns. The suspended transaction context is not passed to any enterprise bean objects or resources that are used by this bean method.

Bean Managed
This value notifies the container that the bean class directly handles transaction demarcation. This property can be specified only for session beans, not for individual bean methods.

Web module

Web application - Distributable

The distributable flag for J2EE Web applications specifies that the Web application is programmed appropriately to be deployed into a distributed servlet container.

Guide

Web applications should be marked as distributable if, and only if, they will be deployed in a WebSphere Application Server cluster or cloned environment.

Web application - Reload interval

Reload interval specifies a time interval, in seconds, in which the Web application file system is scanned for updated files, such as servlet class files or JSPs.

Guide

Reload interval can be defined at different levels for various application components. Generally, the reload interval specifies the time the application server will wait between checks to see if dependent files have been updated and need to be reloaded. Checking file system time stamps is an expensive operation and should be reduced. The default of 3 seconds is good for a test environment because the Web site can be updated without restarting the application server. In production environments, a few times a day is more common.

Web application - Reloading enabled

This specifies whether file reloading is enabled.

Web application - Web components - Load on startup

Indicates whether a servlet is to be loaded at the startup of the Web application. The default is false.

Guide

Many servlets perform resource allocation and other up-front processing in the servlet init() method. These initialization routines can be costly at runtime. By specifying **Load on startup** for these servlets, processing takes place when the application server is started. This avoids runtime delays, which can be encountered on a servlets initial access.

Tuning Java memory

The following section focuses on tuning Java memory. Enterprise applications written in Java involves complex object relationships and utilize large numbers of objects. Although Java automatically manages memory associated with an object's life cycle, it is important to understand the application's usage patterns for objects. In particular, ensure the following:

- The application is not over-utilizing objects
- The application is not leaking objects (that is, memory)
- The Java heap parameters are set to handle the use of objects

Understanding the effect of garbage collection is necessary to apply these management techniques.

The garbage collection bottleneck

Examining Java garbage collection can give insight into how the application is utilizing memory. Garbage collection is a Java strength. By taking the burden of memory management away from the application writer, Java applications are more robust than applications written in languages that do not provide garbage collection. This robustness applies as long as the application is not abusing objects. It is normal for garbage collection to consume anywhere from 5 to 20% of the total execution time of a properly functioning application. If not managed, garbage collection can be one of the biggest bottlenecks for an application, especially when running on SMP server machines.

The garbage collection gauge

Use garbage collection to evaluate application performance health. By monitoring garbage collection during the execution of a fixed workload, users gain insight as to whether the application is over-utilizing objects. Garbage collection can even be used to detect the presence of memory leaks.

Use the garbage collection and heap statistics in Resource Analyzer to evaluate application performance health. By monitoring garbage collection, memory leaks and over-use of objects can be detected.

For this type of investigation, set the minimum and maximum heap sizes to the same value. Choose a representative, repetitive workload that matches production usage as closely as possible (user errors and all). It is also important to allow the application to run several minutes until the application state stabilizes.

To ensure meaningful statistics, run the fixed workload until the state of the application is steady. This usually takes several minutes.

Detecting over-utilization of objects

To see if the application is overusing objects, look in Resource Analyzer at the counters for the JVMPI profiler. The average time between garbage collection calls should be 5 to 6 times the average duration of a single garbage collection. If not, the application is spending more than 15% of its time in garbage collection. Also, look at the numbers of freed, allocated and moved objects.

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

Detecting memory leaks

Memory leaks in Java are a dangerous contributor to garbage collection bottlenecks. They are more damaging than memory overuse because a memory leak ultimately leads to system instability. Over time, garbage collection occurs more frequently until finally the heap is exhausted and Java fails with a fatal `Out of Memory` exception. Memory leaks occur when an unneeded object has references that are never deleted. This most commonly occurs in collection classes, such as Hashtable, because the table itself always has a reference to the object, even after real references have been deleted.

It is a common complaint that applications crash immediately after being deployed in the production environment. High workload is often the cause. This is especially true for leaking applications where the high workload accelerates the magnification of the leakage and a memory allocation failure occurs.

Memory leak testing relates to magnifying numbers. Memory leaks are measured in terms of the amount of bytes or kilobytes that cannot be garbage collected. The delicate task is to differentiate these amounts from the expected sizes of useful and unusable memory. This task is achieved more easily if the numbers are magnified, resulting in larger gaps and easy identification of inconsistencies. The following is a list of important conclusions about memory leaks:

- **Long-running test**
  Memory leak problems manifest only after a period of time, therefore recognizing memory leaks is related to long-running tests. Short runs can lead to false alarms. One of the issues in Java is whether to say that a memory leak is occurring when memory usage has seemingly increased either abruptly or monotonically in a given period. It is possible this kind of increase is valid and the objects created are going to be referenced at a much later time. In other words, how do you differentiate the delayed use of objects from completely unused objects? By running applications long enough, you will get a higher confidence for whether the delayed use of objects is actually occurring. Because of this, memory leak testing cannot be integrated with some other types of tests, such as functional testing, that normally occur early in the process. However, it can be integrated with tests such as stress or durability tests.
- **System test**
  Some memory leak problems occur only when different components of a big project are combined and executed. This is especially true when the interfaces between components produce known or unknown side-effects. The system test is a good opportunity to make these conditions happen.
- **Repetitive test**
  In many cases, memory leak problems occur by successive repetitions of the same test case. The goal of memory leak testing is to establish a big gap between unusable memory and used memory in terms of their relative sizes. By repeating the same scenario over and over again, the number is multiplied in a very progressive way. This is especially helpful if the amount of leaks caused by an execution of a test case is so minimal that it could hardly be noticed in one run.

  Repetitive tests can be used at the system level or module level. The advantage with modular testing is better control. When a module is designed so that everything that takes place within the module is kept private and does not create external side effects including memory usage, testing for memory leaks can be much easier. First, the memory usage before running the module is recorded and then a fixed set of test cases are run repeatedly. At the end of the test run, the current memory usage is repeatedly the one previously recorded and checked, if it has changed significantly. Remember, garbage collection must be forced when recording the actual memory usage. To do this, insert System.gc() in the module where you want garbage collection to occur, or use a profiling tool forces this event to occur.
- **Concurrency test**
  Some memory leak problems can occur only when there are several threads running in the application. Unfortunately, synchronization points are very susceptible to producing memory leaks because of the added complication in the program logic. Careless programming can lead to references being kept or unreleased. The incident of memory leaks is oftentimes facilitated or accelerated by increased concurrency in the system. The most common way to increase concurrency is to increase the number of clients in the test driver.

Consider the following when choosing which test cases to use for memory leak testing:

- A good test case exercises areas of the application where objects are created. Most of the time, knowledge of the application is required. A description of the scenario can suggest creation of data spaces, that is, adding a new record, creating an HTTP session, performing a transaction and searching a record.
- Look at areas where collections of objects are being used. Typically, memory leaks are composed of objects of the same class. Also, collection classes such as Vector and Hashtable are common places where references to objects are implicitly stored by calling corresponding insertion methods. For example, the get method of a Hashtable object does not remove its reference to the object being retrieved.

Resource Analyzer helps to determine if there is a memory leak. For best results, repeat experiments with increasing duration, like 1000, 2000, and 4000-page requests. The Resource Analyzer graph of used memory should have a sawtooth shape. Each drop on the graph corresponds to a garbage collection. There is a memory leak if one of the following occurs:

- The amount of memory used immediately after each garbage collection increases significantly. The sawtooth pattern will look more like a staircase.
- The sawtooth pattern has an irregular shape

Also, look at the difference between the number of objects allocated and the number of objects freed. If the gap between the two increases over time, there is a memory leak.

If heap consumption indicates a possible leak during a heavy workload (the application server is consistently near 100% CPU utilization), yet the heap appears to recover during a subsequent lighter or near-idle workload, this is an indication of heap fragmentation. Heap fragmentation can occur when the JVM is able to free sufficient objects to satisfy memory allocation requests during garbage collection cycles, but does not have the time to compact small free memory areas in the heap into larger contiguous spaces.

Another form of heap fragmentation occurs when small objects (less than 512 bytes) are freed. The objects are freed, but the storage is not recovered, resulting in memory fragmentation.

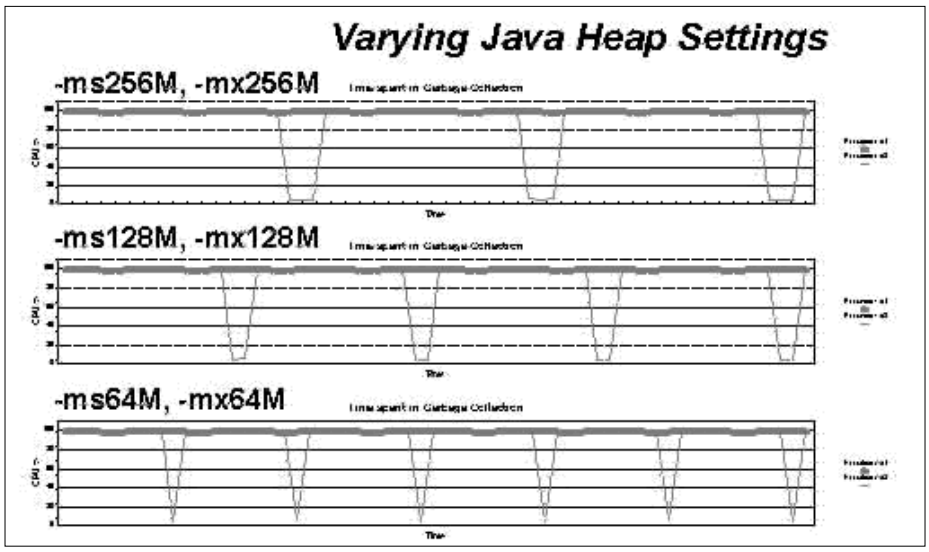
Heap fragmentation can be avoided by turning on the -Xcompactgc flag in the JVM advanced settings command line arguments. The -Xcompactgc ensures that each garbage collection cycle eliminates fragmentation, but with a small performance penalty.

Java heap parameters

The Java heap parameters also influence the behavior of garbage collection. Increasing the heap size allows more objects to be created. Because a large heap takes longer to fill, the application runs longer before a garbage collection occurs. However, a larger heap also takes longer to compact. Garbage collection also takes longer.

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

When tuning a production system where the working set size of the Java application is not understood, a good starting value is to let the initial heap size be 25% of the maximum heap size. The JVM will then try to adapt the size of the heap to the working set size of the application.



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

Run a series of test experiments that vary the Java heap settings. For example, run experiments with 128MB, 192MB, 256MB, and 320MB. During each experiment, monitor the total memory usage. If you expand the heap too aggressively, paging can occur. (Use the **vmstat** command or the Windows NT/2000 Performance Monitor to check for paging.) If paging occurs, reduce the size of the heap or add more memory to the system. When all the runs are finished, compare the following statistics :

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

If the application is not over-utilizing objects and it has no memory leaks, it reaches a state of steady memory utilization, in which garbage collection occurs less frequently and for short duration.

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

Solaris TCP parameters

- **Short description**: Tuning these parameters has a significant performance impact for Solaris. StartupServer.sh sets these:

  ```
  Solaris tcp_time_wait_interval
  Solaris tcp_fin_wait_2_flush_interval
  Solaris tcp_keepalive_interval
  ```

  Many other TCP parameters exist; changing them can affect performance in your environment. For more information about tuning the TCP/IP Stack, see the Web site Tuning your TCP/IP Stack and More.

- **When to try these parameters**: Try these parameters when you are using WebSphere Application Server on Solaris.

  Before the three TCP parameters were changed, the server was stalling during certain peak periods. The **netstat** command showed that many sockets open to port 80 were in the state CLOSE_WAIT or FIN_WAIT_2.

Workload management topology

- **Short description**: WebSphere Application Server provides various Workload Management (WLM) topologies. The following two topologies (name Topology A and B) are examples of workload being sent from one machine:
  - **Topology A** contains a Web server and a WebSphere Application Server plug-in to four machines (each a clone of the other and each containing a Web container and EJB container).

- **Topology B** includes a Web server, a plug-in, and one Web container to four machines (each containing an EJB container).

In both topologies, the Object Request Broker pass-by-reference is selected and the backend database is on its own dedicated machine.

- **When to try adjusting:** Topology A has an advantage because the Web container and EJB container are running in a single JVM. In Topology B, the Object Request Broker pass-by-reference option is ignored between the Web container machine and the EJB container machines. In Topology A, the EJB container uses the same thread passed from the Web container. In other words, the request does not have to be passed from one thread in one JVM to another thread in another JVM.

  Also, if the processor utilization of the four machines is near 100% a fifth machine could be added. Or, if the Web server box is not running at capacity and the Web container processing is not heavy, try freeing the processors on the four machines by moving to the Topology B.
- **Actual benefit:** Throughput for Topology A was 21% better than Topology B. In both cases, the workload was trade on the same hardware: The Web server machine was a RS/6000 (4x450MHz) and the four EJB container machines were each RS/6000 (4x333MHz). None of the machines had a processor utilization more than 85%.
- **Recommended value:** Topology A has an advantage, however, there are many factors related to the application and environment that influences the results.

Number of connections to DB2

- **Recommended value**: When configuring the data source settings for the databases, ensure the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to establish clones, the MaxAppls setting needs to be the maximum number of connections multiplied by the number of clones.

  The same relationship applies to the session manager number of connections. The MaxAppls setting must be at least as high as the number of connections. If you are using the same database for session and data sources, MaxAppls needs to be the sum of the number of connection settings for the session manager and the data sources.
  MaxAppls = (# of connections set for data source + # of connections in session manager) x # of clones

  After calculating the MaxAppls settings for the `WAS` database and each of the application databases, ensure that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls.

  ```
  MaxAgents = sum of MaxAppls for all databases
  ```
- **Related parameters**: See DB2 MaxAppls and DB2 MaxAgents

Individual performance parameters

As mentioned previously, tuning various system components has a strong affect on the performance of WebSphere Application Server. This section discusses how to set the parameters of these individual components in order to bring the system to an optimum level of usage.

Hardware

This section discusses considerations for selecting and configuring the hardware on which the application servers will run.

Processor speed

- **Short description**: Ideally, other bottlenecks have been removed where the processor is waiting on events like input/output and application concurrency. In this case, increasing the processor speed often helps throughput and response times.

System memory

- **Short description**: Increasing memory to prevent the system from paging memory to disk is likely to improve performance.

  Allow at least 512MB memory for each processor.
- **When to try adjusting**: Try adjusting the parameter when the system is paging (and processor utilization is low because of the paging).

Networks

- **Short description**: Run network cards and network switches at full duplex. Running at half duplex decreases performance.

  Verify the network speed can accommodate the required throughput. Make sure that 100MB is in use on 10/100 Ethernet networks.

See the white paper WebSphere Application Server Admin Best Practices for Performance and Scalability for more information regarding hostname resolution on the administrative client host.

Operating system settings

This section discusses considerations for tuning the operating systems in the server environment.

Windows NT/2000 TCP/IP parameters

Windows NT/2000 TcpTimedWaitDelay

- **Short description:** Determines the time that must elapse before TCP can release a closed connection and reuse its resources. This interval between closure and release is known as the TIME_WAIT state or 2MSL (twice the maximum segment lifetime) state. During this time, the connection can be reopened at much less cost to the client and server than establishing a new connection. Reducing the value of this entry allows TCP to release closed connections faster, providing more resources for new connections.
- **When to try adjusting:** If the application running requires rapid release and creation of new connections, and there is a low throughput due to many

connections sitting in TIME_WAIT.

- **How to view or set:**
  1. Using the **regedit** command, access HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\TCPIP\Parameters and create a new REG_DWORD named TcpTimedWaitDelay.
  2. Set the value to decimal 30, which is Hex 0x0000001e.
  3. Restart the system.
  4. By using the **netstat** command, you will be able to see that there are fewer connections in TIME_WAIT.
- **Default value:** 0xF0 (240 seconds = 4 minutes)
- **Recommended value:** The minimum value of 0x1E (30 seconds).
- **Related parameters:** Windows NT/2000 MaxUserPort

Windows NT/2000 MaxUserPort

- **Short description:** Determines the highest port number TCP can assign when an application requests an available user port from the system.
- **How to view or set:**
  1. Using the **regedit** command, access HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\ Services\TCPIP\Parameters and create a new REG_DWORD named MaxUserPort.
  2. Restart the system.
- **Recommended value:** At least decimal 32768.
- **Related parameters:** Windows NT/2000 TCP/IP parameters

Note: These two parameters should be used together when tuning WebSphere Application Server on a Windows NT/2000 operating system.

AIX (4.3.3)

AIX file descriptors (ulimit)

- **Short description**: Specifies the number of open files permitted.
- **When to try adjusting**: The default setting is typically sufficient for most applications. If the value set for this parameter is too low, a `Memory allocation` error is displayed. This error occurs when four clones are run on a S80 24-way and the **ulimit** value needs to be changed to unlimited.
- **How to view or set**: Check the UNIX reference pages on **ulimit** for the syntax for different shells. For the KornShell shell (ksh), to set ulimit to 2000, issue the following command:

  ```
  ulimit -n 2000
  ```

  For large SMP machines with clones, issue the following command:

  ```
  ulimit -unlimited
  ```

  Use the command **ulimit -a** to display the current values for all limitations on system resources.
- **Default value**: For AIX, the default setting is 2000.

Solaris

Solaris file descriptors (ulimit)

- **Short description**: Specifies the number of open files permitted.
- **When to try adjusting**: If the value of this parameter is too low, a `Too many files open` error displays in the WebSphere Application Server stderr.log.
- **How to view or set**: Check the UNIX reference pages on **ulimit** for the syntax for different shells. For KornShell (ksh) the command is:

  ```
  ulimit -n 1024
  ```

  Use **ulimit -a** to display the current values for all limitations on system resources.
- **Default value**: The WebSphere Application Server startupServer.sh script sets this parameter to 1024 if its value is less than 1024.

Solaris TCP_TIME_WAIT_INTERVAL

- **Short description**: This parameter tells TCP how long to keep closed connection control blocks. After the applications complete the TCP connection, the control blocks are kept for the specified time.
- **When to try adjusting**: When high connection rates occur, a large backlog of the TCP connections build up and can slow server performance.

  The server can stall during certain peak periods. If this occurs, The **netstat** command will show that many of the sockets opened to port 80 were in the CLOSE_WAIT or FIN_WAIT_2 state. Visible delays have occurred for up to four minutes, during which the server did not send any responses, but CPU utilization stayed high, with all of the activity in system processes.
- **How to view or set**: Use the **get** command to determine the current interval and the **set** command to specify an interval of 60 seconds. For example:

  ```
  ndd -get /dev/tcp tcp_time_wait_interval
  ndd -set /dev/tcp tcp_time_wait_interval 60000
  ```
- **Default value**: The Solaris default time wait interval is 2400000 milliseconds.
- **Recommended value**: The TCP_TIME_WAIT_INTERVAL parameter can be set as low as 30000 milliseconds. As a starting point, the WebSphere Application Server startupServer.sh script sets it to 60000ms.
- **Related parameters**: See Solaris TCP parameters

Solaris TCP_FIN_WAIT_2_FLUSH_INTERVAL

- **Short description**: The timer interval prohibiting a connection in FIN_WAIT_2 to remain in that state.
- **When to try adjusting**: When high connection rates occur, a large backlog of TCP connections accumulate and can slow server performance.

  The server can stall during peak periods. Using the **netstat** command indicated that many of the sockets opened to port 80 were in CLOSE_WAIT or FIN_WAIT_2 state. Visible delays have occurred for as many as four minutes, during which the server did not send any responses, but CPU utilization stayed high, with all of the activity in system processes.
- **How to view and set**: Use the following commands to determine the current interval or to set the interval to 67.5 seconds:

  ```
  ndd -get /dev/tcp tcp_fin_wait_2_flush_interval
  ndd -set /dev/tcp tcp_fin_wait_2_flush_interval 67500
  ```
- **Default value**: The Solaris default is 675000
- **Recommended value**: 67500
- **Related parameters**: See Solaris TCP parameters

Solaris TCP_KEEPALIVE_INTERVAL

- **Short description**: The timer interval prohibiting an active connection from staying in ESTABLISHED state if one of the peers never responds.
- **When to try adjusting:** If you are concerned with failed communications from clients or peers, this value determines how long a connection will stay open.
- **How to view or set**: Use the following commands to determine the current value or to set the value to 300 seconds:

  ```
  ndd -get /dev/tcp tcp_keepalive_interval
  ndd -set /dev/tcp tcp_keepalive_interval 300000
  ```
- **Default value**: 7200000
- **Recommended Value**: 300000
- **Related parameters**: See Solaris TCP parameters

Other Solaris TCP parameters

Customers have reported success with modifying other Solaris TCP parameters, including the following:

```
tcp_conn_req_max_q
tcp_comm_hash_size
tcp_xmit_hiwat
```

Although significant performance differences have not been seen after raising these settings, the system might benefit.

- **Related parameters**: See Solaris TCP parameters

Solaris kernel semsys:seminfo_semume

- **Short description**: The semsys:seminfo_semume kernel tuning parameter limits the Max Semaphore undo entries per process and needs to be greater than the default (10 on Solaris 7). Because this specifies a maximum value, it does not cause any additional memory to be used unless it is needed.
- **How to view or set**: This value is displayed as SEMUME if the **/usr/sbin/sysdef** command is run. There can possibly be an entry in the /etc/system file for this tuning parameter.

  Set via the /etc/system entry:

  ```
  set semsys:seminfo_semume = 1024
  ```
- **Default value**: 10 on Solaris 7

Solaris kernel semsys:seminfo_semopm

- **How to view or set**: This is displayed SEMOPM if the **/usr/sbin/sysdef** command is run. There can possibly be an entry in the /etc/system file for this tuning parameter.

  Set via the /etc/system entry:

  ```
  semsys:seminfo_semopm = 200
  ```

HP-UX 11

HP-UX 11 settings can be modified to significantly improve WebSphere Application Server performance.

Setting the virtual page size for WebSphere Application Server JVM to 64KB

- **How to view or set**: The command is entered as follows:

  ```
  chatr +pi64M +pd64M /opt/WebSphere/AppServer/java/bin/PA_RISC2.0/native_threads/java
  ```

  The command output provides the current operating system characteristics of the process executable.
- **When to try adjusting:**
- **Default value:**

See the Hewlett Packard Web page for more details about this change.

HP-UX 11 tcp_conn_request_max

- **Short description**: This parameter specifies the maximum number of connection requests that the operating system can queue when the server does not have any available threads. When high connection rates occur, a large backlog of TCP connection requests accumulates and client connections are dropped.
- **When to try adjusting**: This setting can significantly help performance when clients time out while waiting to connect. This situation can be verified by issuing the **netstat -p tcp** command.
  Look for the following value:

  ```
  connect requests dropped due to full queue
  ```
- **How to view or set**: To fix the problem set the TCP_CONN_REQUEST_MAX parameter to 1024 as follows:

  ndd -set /dev/tcp tcp_conn_request_max 1024

  See the following Hewlett Packard Web page for more details about this change.
- **Default value**: The default TCP connection request maximum parameter is 20.

HP-UX 11 Kernel parameter recommendations

- **Short description**: The best results are expected using DB2 or Oracle with the following kernel parameter settings:

Kernel parm	WebSphere Application Server tuning	DB2 tuning	Oracle tuning
maxdsiz	Not adjusted	Not adjusted	Not adjusted
maxdsiz_64b	Not adjusted	Not adjusted	Not adjusted
maxuprc		512	
maxfiles	2,048		
maxfiles_lim	2,048		
nkthreads	10,000		
max_thread_proc	2,048		
nproc		1,028	
nflocks		8,192	
ninode		2,048	
nfile		8,192	
msgseg		32,767	
msgmnb		65,535	
msgmax		65,535	
msgtql		1,024	
msgmap		258	
msgmni		256	
msgssz		16	
semmni		512	70
semmap		514	
semmns		1,024	200
semmnu		1,024	
shmmax		966,367,642	1 GB
shmmseg		16	10
shmmni		300	100

- **When to try adjusting:** See the table above.
- **How to view or set**: Use the HP-UX SAM utility to set the kernel parameters. For directions, see the documentation for the operating system being used.
- **Default value:** See the table above.

Refer to the following Hewlett Packard Web page for more information on HP-UX 11 kernel parameters.

The Web server

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

This section discusses the performance tuning settings associated with the Web servers.

Web server configuration reload interval

- **Short description**: WebSphere Application Server administration tracks a variety of configuration information about WebSphere Application Server resources. Some of this information, such as URIs pointing to WebSphere Application Server resources, also needs to be understood by the Web server. This configuration data is pushed to the Web server through the WebSphere Application Server plug-in at intervals specified by this parameter. Periodic updates allow new servlet definitions to be added without having to restart any of the WebSphere Application Server servers. However, the dynamic regeneration of this configuration information is costly in terms of performance.
- **When to try adjusting**: In a stable production environment.
- **How to view or set**: This parameter, <RefreshInterval=xxxx>, where xxxx is the number of seconds, is specified in the file websphere_root/config/plug-in.xml.
- **Default value:** The default reload interval is 60 seconds.

IBM HTTP Server (IHS) - AIX and Solaris

IHS is a multi-process, single-threaded server. More information see this Web page about Tuning the IBM HTTP Server

MaxClients

- **Short description**: The value of the MaxClients parameter can significantly impact the application, particularly if it is too high. More is not always better. The optimum value depends on the application.
- **How to view or set**: Edit the MaxClients directive in the IBM HTTP Server file httpd.conf. It is located in the directory *IBM HTTP Server_root_directory/conf.*
- **Default value**: 150
- **Related parameters**: See Adjusting WebSphere Application Server system queues

MinSpareServers, MaxSpareServers, and StartServers

- **Short description**: These settings affect the performance of the application. For optimum performance, specify the same value for the MaxSpareServers and the StartServers parameters. Doing this reduces the CPU usage for creating and destroying httpd processes. It pre-allocates and maintains the specified number of processes so that few processes are created and destroyed as the load approaches the specified number of processes (based on MinSpareServers).
- **When to try adjusting:**
- **How to view or set**: Edit the following directives in the file httpd.conf, which is located in the directory *IBM_HTTP_Server_root_directory/conf*:
  - MinSpareServers
  - MaxSpareServers
  - StartServers
- **Default value**:
  - MinSpareServers 5
  - MaxSpareServers 10
  - StartServers 5

iPlanet Web server, Enterprise Edition - AIX and Solaris

The default configuration of the iPlanet Web server, Enterprise Edition provides a single-process, multi-threaded server.

Active threads

- **Short description**: After the server reaches the limit set with this parameter, it will stop servicing new connections until it has finished with the old connections.
- **When to try adjusting**: If this setting is too low, the server can become throttled, resulting in degraded response times.

  To tell if the Web server is being throttled, consult its perfdump statistics. Look at the following data:
  - The **WaitingThreads** count: If this number is getting close to zero, or is zero, the server is not accepting new connections.
  - The **BusyThreads** count: If the WaitingThreads count is close to zero, or is zero, BusyThreads is probably very close to its limit.
  - The **ActiveThreads** count: If this is close to its limit, the server is probably limiting itself.
- **How to view or set**: Use the **Maximum number of simultaneous requests** parameter in the Enterprise Server Manager interface to control the number of active threads within iPlanet Web server, Enterprise Edition. This setting corresponds to the RqThrottle parameter in the magnus.conf file.
- **Default value**: 512

Microsoft Internet Information Server (IIS) - Windows NT/2000

IIS permission properties

- **Short description**: The Web server has several properties that dramatically affect the performance of the application server. The default settings are usually acceptable. However, because it is possible that other products have changed the default settings without user knowledge, make sure to check the IIS settings for the Home Directory permissions of the Web server. The permissions should be set to Script and not to Execute. If the permissions are set to Execute, no

error messages are returned, but the performance of WebSphere Application Server is decreased.

- **How to view or set**: To check or change these permissions perform the following procedure in the Microsoft Management Console:
  1. Select the Web site (usually default Web site). Right-click and select the **Properties** option. Click the **Home Directory** tab.
  2. Set the permissions of the Home Directory: In the **Application** settings ensure that the **Script** check box is checked in the **Permissions** list and that the **Execute** check box is cleared.

  Note: It might be necessary to check the permissions of the sePlugin:
  1. Expand the Web server.
  2. Right-click the **sePlugin** and select **Properties.**
  3. Confirm that the **Execute** permissions are set to Execute.

Number of expected hits per day

- **Short description**: This parameter controls the memory that IIS allocates for connections.
- **How to view or set**: Using the performance window, set this parameter to "More than 100000" in the **Web site properties** panel of the Microsoft Management Console.
- **Default value**: Fewer than 100000.

ListenBackLog parameter

- **Short description**: If using IIS on Windows NT/2000, you are likely to encounter failed connections under heavy load conditions (typically more than 100+ clients). This condition commonly results from IIS rejecting connections.

  Alleviate the condition by using the ListenBackLog parameter to increase the number of requests IIS keeps in its queue.
- **When to try adjusting**: If you intermittently experience a `Netscape unable to locate server` error in a Netscape browser when running a heavy load, consider raising this parameter.
- **How to view or set**:
  1. Use the operating system registry to set the ListenBackLog parameter.
  2. At a DOS command prompt, issue the **regedit** command to access the registry.
  3. In the registry window, locate the parameter in the following directory:
     HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\ *ListenBackLog*
  4. Right-click the parameter to modify it. Adjust the setting according to the server load.
- **Default value**: 25 (decimal)
- **Recommended value**: The ListenBackLog parameter can be set as high as 200 without a negative impact on performance and an improvement in load handling.

MaxPoolThreads, PoolThreadLimit

- **Short description:** MaxPoolThreads controls the number of threads per CPU in the thread pool available for IIS to run Common Gateway Interface (CGI) processes (each process takes one thread). PoolThreadLimit specifies the upper limit for MaxPoolThreads. The default thread limit that IIS can create on a machine is twice the number of MB in RAM on a machine (for example, a server with 512MB of RAM is limited to 1024 threads).
- **How to view or set:** MaxPoolThreads and PoolThreadLimit are set using the following registry entries:
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\MaxPoolThreads
  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\PoolTheadLimit

IBM HTTP Server - Linux

MaxRequestsPerChild

- **Short description**: The MaxRequestsPerChild directive sets the limit on the number of requests that an individual child server process handles. After the number of requests reaches the value set for the MaxRequestsPerChild parameter, the child process dies. The throughput of a simple servlet can improve by 50%.
- **When to try adjusting**:
- **How to view or set**:
  1. Edit the IBM HTTP server file httpd.conf located in the directory IBM_HTTP_Server_root_directory/conf (see the MaxRequestsPerChild directive).
  2. Change the value of the parameter.
  3. Save the changes and restart the IBM HTTP server.
- **Default value**: 30
- **Recommended value**: 500 worked best for a test with a simple servlet.

IBM HTTP Server - Windows NT/2000

The IBM HTTP Server is easily configured. The default settings are usually acceptable.

ThreadsPerChild

- **Short description**: This parameter sets the number of concurrent threads running at any one time within the IBM HTTP Server. Set this value so that it does not bottleneck and just enough traffic is allowed through to the application server.
- **How to view or set**:
  1. Edit the IBM HTTP Server file httpd.conf located in the directory IBM_HTTP_Server_root_directory/conf (see the ThreadsPerChild directive).
  2. Change the value of the parameter.
  3. Save the changes and restart the IBM HTTP server.

How to view thread utilization: There are two ways to find how many threads are being used under load:

1. Use the Windows NT/2000 Performance Monitor:

   To open, click **Start** > **Programs** > **Administrative Tools** > **Performance Monitor**
   In Performance Monitor, click **Edit** > **Add to chart**. Then set the following:
   - Object: IBM HTTP Server
   - Instance: Apache
   - Counter: Waiting for connection
   - To calculate the number of busy threads, subtract the number waiting (Windows NT/2000 Performance Monitor) from the total available (ThreadsPerChild).
2. Use IBM HTTP Server server-status (this choice works on all platforms, not just Windows NT/2000)

   Follow these steps to use IBM HTTP Server server-status:
   1. Edit the IBM HTTP Server file httpd.conf as follows:
      - Remove the comment character "#" from the following lines:
        - #LoadModule status_module modules/ApacheModuleStatus.dll
        - #<Location /server-status>
        - #SetHandler server-status
        - #</Location>
   2. Save the changes and restart the IBM HTTP server.
   3. In a Web browser, go to the following URL and click **Reload** to update status: http://yourhost/server-status.
      Alternatively, if the browser supports refresh, go to http://yourhost/server-status?refresh=5 to refresh every 5 seconds. You will see (along with additional information): 5 requests currently being processed, 45 idle servers.

- **Default value**: 50 (for IBM HTTP Server 1.3.19).
- **Related parameters**: See Adjusting WebSphere Application Server system queues

ListenBackLog

- **Short description**: When several clients request connections to the IBM HTTP Server, and all threads (see ThreadsPerChild) are being used, there is a queue to hold additional client requests. The ListenBackLog directive sets the length of this pending connections queue. However, if you are using the default Fast Response Cache Accelerator (FRCA) feature, the ListenBackLog directive is not used because FRCA has its own internal queue.
- **How to view or set**: For non-FRCA:
  1. Edit IBM HTTP Server file httpd.conf.
  2. Add or view the ListenBackLog directive.
- **Default value**: For HTTP Server 1.3.19:
  1024 with FRCA enabled
  511 with FRCA disabled (the default)
- **Recommended value**: Use the defaults.

The WebSphere Application Server process

Each WebSphere Application Server process has several parameters influencing application performance. Each application server in the WebSphere Application Server product comprises an EJB container and a Web container.

Use the WebSphere Application Server administrative console to configure and tune applications, Web containers, EJB containers, application servers and nodes in the administrative domain.

Adjusting the operating system priority of the WebSphere Application Server process

- **Short description:** Improving the operating system process priority of the application server can help performance. On UNIX systems, a smaller setting results in the process having a higher priority. For more information about process priorities, refer to the following publications:
  - For AIX, refer to the AIX Performance Tuning Guide, Versions 3.2 and 4
  - Refer to the Hewlett Packard Web page for more information about improving a Hewlett Packard system.
- **When to try adjusting:** Try adjusting the priority at any time.
- **Actual benefit:** By improving the priority of an application server, the performance team has experienced improvements up to 5% on AIX. Improvement has also been seen on Solaris and HP.
- **How to view or set:**
  1. In the administrative console, select the application server you are tuning.
  2. Click the **Advanced** tab.
  3. Specify the value in the **Process Priority** field and click **Apply**.
  4. Save the changes and restart the application server.

  How to see parameter utilization: On UNIX, use the command **ps -efl** to see the current process priority.
- **Default value:** 20
- **Recommended value:** 0

Web containers

To route servlet requests from the Web server to the Web containers, the product establishes a transport queue between the Web server plug-in and each Web

container.

Web container maximum thread size

- **Short description**: Use the maximum thread size parameter to specify the maximum number of threads that can be pooled to handle requests sent to the Web container. Requests are sent to the Web container through any of the HTTP transports.
- **How to view or set**:
  1. In the administrative console, select the application server you are tuning and then click the **Services** tab.
  2. Click **Web Container Service** and then click **Edit Properties**.
  3. In the **Web Container Service** window, click the **General** tab.
  4. Specify the value in the **Maximum Thread Size** field.
  5. Return to the **Services** panel and click **Apply** to ensure the changes are saved.
  6. Stop and restart the application server.

  Resource Analyzer displays a metric called Percent Maxed that determines the amount of time that the configured threads are in use. If this value is consistently in the double-digits, then the Web container could be a bottleneck and the number of threads should be increased.
- **Default value**: 50
  Note: For Linux systems, the recommended value is 25.
- **Related parameters**: See Adjusting WebSphere Application Server system queues and Prepared statement cache size

Web container transport maximum Keep-Alive

- **Short description:** This is the maximum number of concurrent connections to the Web container that are allowed to be kept alive (that is, to be processed in multiple requests). The Web server plug-in keeps connections open to the application server as long as it can. However, if the value of this property is too small, performance is negatively impacted because the plug-in has to open a new connection for each request instead of sending multiple requests through one connection. The application server might not accept a new connection under a heavy load if there are too many sockets in TIME_WAIT state. If all client requests are going through the Web server plug-in and there are many TIME_WAIT state sockets for port 9080, the application server is closing connections prematurely and decreasing performance. The application server will close the connection from the plug-in (or from any client) for any of the following reasons:
  1. The client request was an HTTP 1.0 request when the Web server plug-in always sends HTTP 1.1 requests.
  2. The maximum number of concurrent Keep-Alives was reached. A Keep-Alive must be obtained only once for the life of a connection, that is, after the first request is completed but before the second request can be read.
  3. The maximum number of requests for a connection was reached. This helps prevent denial of service attacks in which a client tries to hold on to a Keep-Alive connection forever.
  4. A time out occurred while waiting to read the next request or to read the remainder of the current request.
- **How to view or set:**
  1. Find the **Web container service** panel on the administrative console.
  2. Select the **Transport** tab and set the **Maximum Keep-Alive** value.
- **Default value:** The default value is 45, which is 90% of the default maximum number of threads in the Web container thread pool.
- **Recommended value:** The value should be at least 90% of the maximum number of threads in the Web container thread pool. If it is 100% of the maximum number of threads in the Web container thread pool, all the threads could be consumed by Keep-Alive connections leaving no threads available to process new connections.

Web container transport maximum requests per Keep-Alive

- **Short description:** The maximum number of requests allowed on a single Keep-Alive connection. This helps prevent denial of service attacks when a client tries to hold on to a Keep-Alive connection. The Web server plug-in keeps connections open to the application server as long as it can, providing optimum performance.
- **How to view or set:**
  1. Find the **Web container service** panel on the administrative console.
  2. Select the **Transport** tab and set the **Maximum Keep-Alive** value.
- **Default value:** The default value is 100.
- **Recommended value:** If application server requests are received from the plug-in only, increase the value to improve performance.

URL invocation cache

- **Short description**: The invocation cache holds information for mapping request URLs to servlet resources.

  A cache of the requested size is created for each thread. The number of threads is determined by the Web container maximum thread size setting.

  Note: A larger cache uses more of the Java heap, so you might need to increase maximum Java heap size. For example, if each cache entry requires 2KB, maximum thread size is set to 25, and the URL invocation cache size is 100; then 5MB of Java heap are required.
- **When to try adjusting**: If more than 50 unique URLs are actively being used (each JSP is a unique URL), increase this parameter.
- **How to view or set**: The size of the cache can be specified for the application server along with other JDK parameters by:
  1. In the administrative console, click the application server you are tuning.
  2. Click the **JVM Setting** tab.
  3. On the same panel, click **Add** in the **System Properties** section.
  4. Add the name invocationCacheSize and a value of 50.
  5. Click **Apply** to ensure that the changes are saved.
  6. Stop and restart the application server.
- **Default value**: 50

- **Related parameters**: See Heap size settings

Allow thread allocation beyond maximum

- **Short description:** When this option is selected, more Web container threads can be allocated than specified in the maximum thread size field.
- **How to view or set:**
  1. In the administrative console, select the application server you are tuning and then click the **Services** tab.
  2. Click **Web Container Service** and then click **Edit Properties**.
  3. In the **Web Container Service** window, click the **General** tab.
  4. For a new setting, click on the checkbox.
  5. Return to the **Services** panel and click **Apply** to ensure the changes are saved.
  6. Stop and restart the application server.
- **Default value:** The default value is for the setting to be "unchecked" (the thread pool can not grow beyond the value specified for the maximum thread size).
- **Recommended value:** This option is intended to handle brief loads beyond the configured maximum thread size. However, use caution when selecting this option because too many threads can cause the system to be overloaded.

EJB container

Cache settings

- **Short description**: To determine the cache absolute limit, multiply the number of enterprise beans active in any given transaction by the total number of concurrent transactions expected. Then add the number of active session bean instances.

  Use the Resource Analyzer to view bean performance information.
- **How to view or set**: Edit the EJB container service properties for the application server you are tuning.
- **Default value**:
  Cache size = 2047
  Cache preferred limit = 2000
  Cache clean-up interval = 1000

Break CMP enterprise beans into several enterprise bean modules

- **Short description**: The load time for hundreds of beans can be improved by distributing the beans across several JAR files and packaging them to an EAR file. This is faster when the administrative server attempts to start the beans(8-10 minutes versus more than one hour when one JAR file is used).

Security

This section discusses how various settings related to security affect performance.

Disabling security

- **Short description**: Security is a global setting. When security is enabled, performance can be decreased between 10-20%. Therefore, it is practical to disable security when it is not needed.
- **How to view or set**: In the administrative console, click **Console** > **Security Center**.
- **Default value**: By default, security is not enabled.

Fine-tune the security cache time out for the environment

- **Short description**: If WebSphere Application Server security is enabled, the security cache time out can influence performance. The time out specifies how often to refresh the security-related caches.

  Security information pertaining to beans, permissions, and credentials is cached. When the cache time out expires, all cached information becomes invalid. Subsequent requests for the information result in a database lookup. Sometimes, acquiring the information requires invoking an Lightweight Directory Access Protocol(LDAP)-bind or native authentication, both of which are relatively costly operations for performance.

  Experiment to find the best trade-off for the application, based on the usage patterns and security needs for the site.
- **Actual benefit observed**: In a simple 20-minute performance run, when the cache time out was set so that a time out did not occur, a 40% performance improvement was achieved.
- **How to view or set**: In the administrative console, click **Console** > **Security Center General** > **SecurityCacheTimeout**
- **Default value:** The default is 600.

Security cache types and sizes (system parameters)

The following system properties determine the initial size of the cache primary and secondary Hashtables, which affect the frequency of rehashing and the distribution of the hash algorithms. The larger the number of available hash values, the less likely a hash collision will occur, and more likely a slower retrieval time. If several entries compose a cache Hashtable, creating the table in a larger capacity allows the entries to be inserted more efficiently rather than letting automatic rehashing determine the growth of the table. Rehashing causes every entry to be moved each time it is done.

com.ibm.websphere.security.util.LTPAAuthCacheSize

- **Short description:** This cache stores basic authentication credentials at the Security Server. Whenever an Lightweight Third Party Authentication (LTPA) token expires, a new token is generated from the basic authorization credentials in this cache. If no basic authorization credentials exist, the requesting browser must send the basic authorization credentials to the Security Server. The browser will prompt the user for userID and password if no cookie exists containing the credentials.

com.ibm.websphere.security.util.LTPATokenCacheSize

- **Short description:** This cache stores LTPA credentials in the cache using the LTPA token as a lookup value. When using an LTPA token to log in, the LTPA credential is created at the Security Server for the first time. This cache prevents the need to go to the Security Server on subsequent logins using an LTPA token.

com.ibm.websphere.security.util.CredentialCacheSize

- **Short description:** Given the userID and password for login, this cache returns the concrete credential object, either Local OS or LTPA, without the need to repeat authentication at the Security Server. If the credential object has expired, the need to repeat authentication is required.

com.ibm.websphere.security.util.LTPAValidationCacheSize

- **Short description:** Given the credential token for login, this cache returns the concrete LTPA credential object, without the need to revalidate at the Security Server. If the token has expired, the need to revalidate is required.

com.ibm.websphere.security.util.PermissionCacheSize

- **Short description:** This cache holds the WebSphere Application Server permission objects retrieved when a getGrantedPermissions method is called. If access to the same resource by the same principal occurs again, the permissions will be retrieved rapidly from the cache instead of going to the repository on the administrative server. This cache is common to both enterprise bean and Web-granted permissions.

com.ibm.websphere.security.util.AdminBeanCacheSize

- **Short description:** Stores information, including the required permissions, about enterprise beans that have been deployed in the administrative server.

com.ibm.websphere.security.util.BeanCacheSize

- **Short description:** Stores information, including the required permissions and RunAs mode, about enterprise beans that have been deployed in a container on the application server.

Configure SSL sessions appropriately

- **Short description**: The SSLV3Timeout value specifies the time interval after which SSL sessions are renegotiated. This is a high setting, and modification probably does not provide any significant impact. By default, it is set to 9600 seconds.

  The Secure Association Service (SAS) feature establishes an SSL connection only if it goes out of the JVM (to another JVM). Therefore, if all the beans are co-located within the same JVM, the SSL used by SAS is not expected to hinder performance.
- **How to view or set**: Modify the SSLV3Timeout and other SAS properties by editing sas.server.props and sas.client.props files. The files are located in the *product_installation_root*\properties directory, where *product_installation_root* is the directory where WebSphere Application Server is installed.
- **Default value:** The default is 9600 seconds.

Object Request Broker (ORB)

Several settings are available for controlling internal ORB processing. Use these to improve application performance in the case of applications containing enterprise beans.

Use the **Services** tab and then **Edit Properties** for Object Request Broker, for the default server or any additional application server configured in the administrative domain, to set these parameters.

Pass-by-value versus pass-by-reference (NoLocalCopies)

- **Short description**: The EJB 1.1 specification states that method calls are to be pass-by-value. Pass-by-value means that for every remote method call, the parameters are copied onto the stack before the call is made. This can be expensive. It is possible to specify pass-by-reference, which passes the original object reference without making a copy of the object.
- **Actual benefit observed**: If the EJB client and EJB server are installed in the same WebSphere Application Server instance, specifying pass-by-reference can improve performance up to 50%.
- **When to try adjusting**: Pass-by-reference helps performance only in the case where non-primitive object types are being passed as parameters. Therefore, int and floats are always copied, regardless of the call model.

  WARNING: Pass-by-reference can be dangerous and can lead to unexpected results. If an object reference is modified by the remote method, the change might be seen by the caller.
- **How to view or set**:
  1. Click the **Services** tab in the application server you are tuning of the Object Request Broker.
  2. Click the **Advanced** tab.
  3. Check the **Pass by Reference** and click **OK**.
  4. Click **Apply** to save the changes.
  5. Stop and restart the application server.
- **Default value**: Pass-by-value

If the application server expects a large workload for enterprise bean requests, the ORB configuration is critical. Take note of the following properties:

com.ibm.CORBA.ServerSocketQueueDepth

- **Short description:** This property corresponds to the length of the TCP/IP stack listen queue. and prevents WebSphere Application Server from rejecting requests when there is not space in the listen queue.
- **When to try adjusting:** If there are many simultaneous clients connecting to the server-side Object Request Broker, this parameter can be increased to support the heavy load (up to 1000 clients).
- **How to view or set:** To set the property, follow these steps:

1. In the administrative console, click the application server you are tuning.
2. Click the **JVM Settings** tab and then click the **Advanced JVM Settings** option (you might have to scroll down to find the Advanced JVM Settings).
3. In the **Command Line Arguments** field type
   -Dcom.ibm.CORBA.ServerSocketQueueDepth=200.

- **Default value:** The default value is 50.

com.ibm.CORBA.MaxOpenConnections and Object Request Broker connection cache maximum

- **Short description:** This property has two names and corresponds to the size of the ORB connection table. It sets the standard for the number of simultaneous ORB connections that can be processed.
- **When to try adjusting:**If there are many simultaneous clients connecting to the server-side Object Request Broker, this parameter can be increased to support the heavy load (up to 1000 clients).
- **How to view or set:** This property can be set by a specialized field or through the JVM Command Line Arguments field. The specialized field has a limit of 256, therefore, if the parameter is set to a higher value, use the JVM Command Line Arguments field. To use the specialized field, follow these steps:
  1. Click the **Services** tab in the application server you are tuning of the Object Request Broker.
  2. Select the **General** tab.
  3. Update the Connection Cache Maximum field and click **OK**.
  4. Click **Apply** to save the changes.
  5. Restart the application server.

  Follow these steps to set the property through the JVM Command Line Arguments field:
  1. In the administrative console, select the application server that you are tuning.
  2. Click the **JVM Settings** tab and then click the **Advanced JVM Settings** option (you might have to scroll down to find the Advanced JVM Settings).
  3. In the **Command Line Arguments** field, type
     -Dcom.ibm.CORBA.MaxOpenConnections=800.
- **Default value:** The default is 240. .

Object Request Broker thread pool size

- **Short description**: This property relates to the size of the ORB thread pool. A worker thread is taken from the pool to process requests from a given connection.
- **When to try adjusting:** If there are many simultaneous clients connecting to the server-side Object Request Broker, this parameter can be increased to support the heavy load (up to 1000 clients).
- **How to view or set**:
  1. In the administrative console, select the application server you are tuning and then click the **Services** tab.
  2. Select **Object Request Broker** and then **Edit Properties**.
  3. The thread pool size is on the **General Properties** panel.
- **Related parameters**: See Adjusting WebSphere Application Server system queues
- **Default value:** 20

Using JNI ReaderManager and ReaderThreads

- **Short description:** The ORB has a set of threads called ReaderThreads that perform actual input/output through socket streams. A ReaderManager thread assigns a reader thread for every IIOP Connection established by WebSphere Application Server and its clients. By default, a new thread is created for every Internet Inter-Orb Protocol (IIOP) connection established. That thread is destroyed when the connection is closed. There is an option to use a different type of reader thread called the JNIReader, in which multiple connections are assigned to the reader. Thus, a JNIReader thread multiplexes on a set of sockets. The corresponding manager thread, called the JNIReaderManager, takes care of assigning connections to the JNIReader threads. It uses a round-robin algorithm.

  Using JNIReaders requires less memory because a fixed set of threads must be created. It saves time because thread creations are done only once during initialization and threads are never destroyed. The JNIReader is a C-native implementation and should be faster than the default reader thread.
- **When to try adjusting:** JNIReaders are highly recommended for applications with heavy enterprise bean workloads (200 to 1000 or more enterprise bean clients, including servlets that call enterprise bean methods).
- **How to view or set:** To use the JNIReaders, set Object Request Broker properties. This can be done for both the administrative server and application server. To set it in the application, follow these steps:
  1. Select the **JVM Settings** tab.
  2. Click the **Advanced JVM Settings**.
  3. In the **Command Line Arguments** field, enter the following:
     -Dcom.ibm.CORBA.numJNIReaders=xx
     -Dcom.ibm.CORBA.ReaderManagerImpl=com.ibm.CORBA.iiop.JNIReaderManager
     Note: xx is the number of JNIReader threads you want to allocate. There is no absolute number to choose, but a common guideline is to equate this to the number of processors available in the system, in addition to 1. (xx=the number of processors available in the system + 1)
  4. Click **OK** and then click **Apply**.

  To set it in the administrative server, follow these steps:
  1. Edit the admin.config file and add the same items in step 3 above to com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs
  2. Save the changes and and restart the administrative server.

WARNING: Make sure that the JNI library implementation of the JNIReader is in the WebSphere Application Server bin directory. For Intel platform, the library is Selector.dll and for UNIX, it is libSelector.a or libSelector.so. For Unix, if the prefix "lib" is missing, the file should be renamed.

Java Virtual Machines (JVMs)

Tuning the JVM

The JVM offers several tuning parameters affecting the performance of WebSphere Application Servers (which are primarily Java applications), as well as the performance of your applications.

Sun JDK 1.3 HotSpot -server warmup

- **Short description:** The HotSpot JVM introduces adaptive JVM technology containing algorithms for optimizing byte code execution over time. The JVM runs in two modes, -server and -client. Performance can be significantly enhanced if running in -server mode and a sufficient amount of time is allowed for a HotSpot JVM to warmup by performing continuous execution of byte code.
- **When to try adjusting:** In most cases, -server mode should be run. This produces more efficient runtime execution over extended periods. The -client option can be used if a faster startup time and smaller memory footprint are preferred, at the cost of lower extended performance.
- **How to view or set:** In WebSphere Application Server Version 4.0, the -server option is enabled by default. Follow these steps to change to change the -client mode:
  1. In the administrative console, select the application server you are tuning.
  2. Click **JVM Settings** and then **System Properties**.
  3. In **System Properties** add the name-pair: HotSpotOption / client.
- **Recommended value:** -server

Sun JDK 1.3 HotSpot new generation pool size

- **Short description:** Most garbage collection algorithms approach the problem by iterating every object in the heap to determine which ones can be freed. The HotSpot JVM introduces generational garbage collection which makes use of separate memory pools to contain objects of different ages. These pools can be garbage collected independently from one another. The sizes of these memory pools can be adjusted. Extra work can be avoided by sizing the memory pools so that short-lived objects will never live through more than one garbage collection cycle.
- **When to try adjusting:** If garbage collection has become a bottleneck, try customizing the generation pool settings.
- **How to view or set:**
  1. In the administrative console select the application server you are tuning.
  2. Click **JVM Settings** > **Advanced JVM Settings** > **Command line arguments**. The following are the values that are set:
     -XX:NewSize (lower bound)
     -XX:MaxNewSize (upper bound).
- **Recommended value:** Bound the new generation between 25 to 50% the total heap size.

Just In Time (JIT) compiler

- **Short description**: The Just In Time (JIT) compiler can significantly affect performance.
- **When to try adjusting**: In all cases, run with JIT enabled, which is the default.

Heap size settings

- **Short description**: These are used to set the maximum and initial and heap sizes for the JVM.

  In general, increasing the size of the Java heap improves throughput until the heap no longer resides in physical memory. After the heap begins swapping to disk, Java performance drastically suffers. Therefore, the maximum heap size needs to be low enough to contain the heap within physical memory.

  The physical memory usage must be shared between the JVM and the other applications, for instance, the database. For assurance, use a smaller heap (for example 64MB, on machines with less memory).

  Try a maximum heap of 128MB on a smaller machine (that is , less than 1GB of physical memory), 256MB for systems with 2GB memory, and 512MB for larger systems. The starting point depends on the application.

  If performance runs are being conducted and highly repeatable results are needed, set the initial and maximum sizes to the same value. This eliminates any heap growth during the run. For production systems where the working set size of the Java applications is not well understood, an initial setting of one-fourth the maximum setting is a good starting value. The JVM will then try to adapt the size of the heap to the working set of the Java application.
- **How to view or set**: On the JVM settings tab of the application server:
  - Initial java heap size
  - Maximum java heap size

Class garbage collection

- **Short description**: In most cases, run with class garbage collection turned on. This is the default.
- **How to view or set**: Disabling class garbage collection enables more class reuse, which, in some cases, has resulted in small performance improvements.

  Use the command line property of the default server or any additional application server you configure in the administrative domain to set the JVM parameters:
  1. In the administrative console, click the application server you are tuning.
  2. Click the **Services** tab.
  3. In the **Advanced** tab, click **JVM Settings**.
  4. In **Command Line Arguments** enter -Xnoclassgc.

The database

WebSphere Application Server is tightly integrated with a supported database of your choice. For information about supported database products, see the product prerequisites Web site at www.ibm.com/software/webservers/appserv/library.html. WebSphere Application Server uses the database as a persistent backing store for administration, as well as to store session state and enterprise bean data for the application.

If the application uses WebSphere Application Server session state, JDBC database connection pooling or enterprise beans, pay special attention to how these resources and their database settings are configured within the administrative domain. During WebSphere Application Server installation, a database named `WASnn` is established, where nn is the release identifier, although a different name can be used. This document assumes `WAS40` is used.

Database location

- **Short description**: The DB2 database can be located on the same machine as the WebSphere Application Server or on a remote machine, although, in a high input/output OLTP application, the operating system, DB2 logs, tables and indexes should have their own disk space for sufficient input/output parallelism.
- **When to try adjusting**: If the machine is experiencing a bottleneck because its processing power is being used (CPU constrained), significant improvements can be obtained by installing the database to a separate machine.

  For scalability, it is likely the database is established on a separate machine, particularly if clustering is used. This relates to the WebSphere Application Server database, any application database, and the WebSphere Application Server session database (if persistent session is used).
- **Default value**: The default installation places the database on the local machine.

WebSphere Application Server data source connection pool size

- **Short description**: When accessing any database, the initial database connection is an expensive operation. WebSphere Application Server supports JDBC 2.0 Standard Extension APIs to provide support for connection pooling and connection reuse. The connection pool is used for direct JDBC calls within the application, as well as for enterprise beans using the database.

  If clones are used, one data source pool exists for each clone. This is important when configuring the database server maximum connections.
- **Actual benefit observed**: When the connection pooling capabilities are used, performance improvements up to 20 times the normal results can be realized.
- **How to view or set**: Use the administrative console to specify the database connection pool options.
- **Default value**: Minimum connection pool size: 1, Maximum connection pool size: 10
- **How to view parameter utilization**: Enable the Resource Analyzer counter collection:
  1. Start the Resource Analyzer
  2. Select the application server you are tuning.
  3. Expand the database connection pools category
  4. Select the data source of interest.
  5. Click the data source and set to high. High is necessary to obtain the percentUsed counter.
  6. Click **Run** to start collecting statistics, and observe the following:
     - Pool size = average number of connections in the pool to the database
     - Percent used = average percent of the pool connections in use
     - Concurrent waiters = average number of threads waiting for a connection

     Use Resource Analyzer to find the optimal number of pool connections that can reduce values for these numbers. If Percent Used is consistently low, consider decreasing the number of connections in the pool.
- **Recommended value**: Better performance is generally achieved if the value for the connection pool size is set lower than the value for the Max Connections in the Web container. It has been found that lower settings (10-30 connections) perform better than higher (more than 100) settings.

  On UNIX platforms, a separate DB2 process is created for each connection. This quickly affects performance on systems with low memory, and errors can occur.

  Each Entity EJB transaction requires an additional connection to the database specifically to handle the transaction. Be sure to take this into account when calculating the number of data source connections.

  Deadlock can occur if the application requires more than one concurrent connection per thread, **and** the database connection pool is not large enough for the number of threads. Suppose each of the application threads requires two concurrent database connections and the number of threads is equal to the maximum connection pool size. Deadlock can occur when both of the following are true:
  - Each thread has its first database connection, and all are in use.
  - Each thread is waiting for a second database connection, and none would becomes available because all threads are blocked.

  To prevent the deadlock in this case, the value set for the database connection pool must be at least one higher, so that at least one of the waiting threads can complete its second database connection, freeing up database connections.

  To avoid deadlock, code the application to use, at most, one connection per thread. If the application is coded to require `C` concurrent database connections per thread, the connection pool must support at least the following number of connections, where `T` is the maximum number of threads.

  ```
  T * (C - 1) + 1
  ```

  The connection pool settings are directly related to the number of connections that the database server is configured to support. If the maximum number of connections in the pool is raised, and the corresponding settings in the database are not raised, the application fails and SQL exception errors are displayed in the stderr.log file.

- **Related parameters**: See Adjusting WebSphere Application Server system queues, Prepared statement cache size, and Number of connections to DB2

Prepared statement cache size

- **Short description**: The WebSphere Application Server data source optimizes the processing of prepared statements. A prepared statement is a precompiled SQL statement that is stored in a prepared statement object. This object is used to efficiently execute the given SQL statement multiple times.

Note: Prepared statements are optimized for handling parametric SQL statements that benefit from precompilation. If the JDBC driver specified in the data source supports precompilation, the creation of the prepared statement will send the statement to the database for precompilation. Some drivers might not support precompilation. In this case, the statement might not be sent to the database until the prepared statement is executed.

- **Actual benefit observed**: In test applications, tuning the prepared statement cache improved throughput by 10-20%.
- **How to view or set**: In the administrative console click **Resources** > **JDBC Properties** > **Database Driver** > **Data source** > **Connection Pooling** > **Statement Cache Size**. The **Statement Cache Size** field should contain a value that is the total cache size, not the size per container.
- **Recommended value**: If the cache is not large enough, useful entries will be discarded to make room for new entries. In general, the more prepared statements your application has, the larger the cache should be. For example, if the application has 5 SQL statements, and the data source maximum connection is 10, set the prepared statement cache size to 50.

  Resource Analyzer can help tune this setting to minimize cache discards. Use a standard workload that represents a typical number of incoming client requests, use a fixed number of iterations, and use a standard set of configuration settings.

  Follow these instructions to use the Resource Analyzer:

  1. Start Resource Analyzer.
  2. Click **Database Connections** > **Monitoring Settings**.
  3. Before starting the benchmark workload, right-click **Database Connections, Clear Values** and **Reset to Zero** (if necessary), and **Start**.
  4. Start the workload and run to completion. After the workload finishes, record the value reported for PrepStmt Cache Discards.
  5. Stop the application server and make the following adjustments: Click **Data Source** > **Connection Pooling** > **Statement Cache Size value**.
  6. Rerun the workload and record the Resource Analyzer value reported for PrepStmt Cache Discards.

  The best value for **Data Source** > **Connection Pooling** > **Statement Cache Size** is the setting used to get either a value of zero or the lowest value for PrepStmt Cache Discards. This indicates the most efficient number for a typical workload.
- **Related parameters:** See Adjusting WebSphere Application Server system queues, Max Connections, and WebSphere Application Server data source connection pool size

Other JDBC parameters
In addition to setting the prepared statement cache size, you can set other specific properties for JDBC drivers. For example, using Oracle, you can increase the number of rows to fetch while getting result sets with the following statement:
name="defaultRowPrefetch", value="25"
Enter these types of custom properties on the **General** tab for the database.

DB2

DB2 has many parameters that can be configured to optimize database performance. For complete DB2 tuning information, refer to the DB2 UDB Administration Guide: Performance.

Use TCP sockets for DB2 on Linux

- **Short description**: On Linux platforms, whether the DB2 server resides on a local machine with WebSphere Application Server or on a remote machine, configure the DB2 application databases to use TCP sockets for communications with the database.
- **How to view or set**: The directions for configuring DB2 on Linux can be found in the WebSphere Application Server installation documentation for the various operating systems. It includes specifics for setting DB2COMM for TCP/IP and corresponding changes required in the etc/service file.
- **Default value**: Shared memory for local databases
- **Recommended value**: On Linux, change the specification for the DB2 application databases and any session databases from shared memory to TCP sockets. Even the administrative repository, typically a DB2 database named `WAS40`, must go through TCP.

DB2 MaxAppls

- **Related parameters**: See Number of connections to DB2

DB2 MaxAgents

- **Related parameters**: See Number of connections to DB2

DB2 buffpage

- **Short description**: Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. The purpose of the buffer pool is to improve database system performance. Data can be accessed much faster from memory than from disk.
- **How to view or set**: To view the current value of buffpage for database x, issue the DB2 command **get db cfg for x** and look for the value of BUFFPAGE.

  To set BUFFPAGE to a value of n, issue the DB2 command **update db cfg for x using BUFFPAGE n** and be sure NPAGES is -1 as follows:

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

- **How to view parameter utilization:** Collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio.
  1. Collect the snapshot by issuing the following DB2 commands:
     - **update monitor switches using bufferpool on**

    - **get monitor switches** (To see that bufferpool monitoring is on)
    - **reset monitor all** (To clear monitor counters)
2. Run the application.
3. Issue the following commands:
    - **get snapshot for all databases** (Issue this command before all applications disconnect from the database, otherwise statistics will be lost.)
    - **update monitor switches using bufferpool off** (Make sure not to forget this step!)
4. To calculate the hit ratio, look at the following snapshot statistics for the database:
    - Buffer pool data logical reads
    - Buffer pool data physical reads
    - Buffer pool index logical reads
    - Buffer pool index physical reads
5. The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk in order to service a page request. That is, the page was already in the buffer pool. The greater the buffer pool hit ration, the lower the frequency of disk input/output. The buffer pool hit ratio can be calculated as follows:
    - P = buffer pool data physical reads + buffer pool index physical reads
    - L = buffer pool data logical reads + buffer pool index logical reads
    - Hit ratio = (1-(P/L)) * 100%

DB2 query optimization level

- **Short description**: When a database query is executed in DB2, various methods are used to calculate the most efficient access plan. The query optimization level parameter sets the amount of work and resources that DB2 puts into optimizing the access plan. The range is from zero to 9.

  An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.

  For more information, refer to the DB2 documentation and the IBM DB2 Web site.
- **How to view or set**: The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the dft_queryopt parameter. Dynamic SQL statements use the optimization class specified by the current query optimization special register which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:
  Set current query optimization = 1
  If the current query optimization register has not been set, dynamic statements will be bound using the default query optimization class.
- **Default value**: 5
- **Recommended value**: Set the optimization level for the needs of the application. Use high levels should only when there are very complicated queries.

DB2 reorgchk

- **Short description**: The performance of the SQL statements can be impaired after many updates, deletes or inserts have been made. Performance can be improved by obtaining the current statistics for the data and rebinding.
- **How to view or set**: Use the following DB2 command to issue runstats on all user and system tables for the database you are currently connected to:

  ```
  reorgchk update statistics on table all
  ```

  You should then rebind packages using the **bind** command.

  In order to see if runstats has been done, issue the following command on DB2 CLP:
  db2 -v "select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes"

  If no runstats has been done, nleaf and nlevels will be filled with -1 and stats_time will have an empty entry "-".

  If runstats was done already, the real-time stamp when the runstats was completed will also be displayed under stats_time. If you think the time shown for the previous runstats is too old, do runstats again.

DB2 MinCommit

- **Short description:** This parameter allows you to delay the writing of log records to a disk until a minimum number of commits have been performed, reducing the database manager overhead associated with writing log records. For example, if MinCommit is set to 2, a second commit would cause output to the transaction log for the first and second commits. The exception is when a one-second time out forces the first commit to be output if a second commit does not come along within one second.
- **Actual benefit:** For Trade EJB_ALT, 90% of the disk input/output was related to the DB2 transaction log. Changing MinCommit from 1 to 2 cut the number of disk input/output by 50%. Setting MinCommit to 4 resulted in 33% the number of disk input/output (compared to MinCommit of 1). Processor utilization on the DB2 machine was reduced by 35% (started at 20% busy) and disk input/output wait decreased by 50% (started at 10%). For this particular setup (DB2 machine was not very busy to start with) it is worth noting that the overall throughput and response did not improve.
- **When to try adjusting:**Try to adjust this parameter if the disk input/output wait is more than 5% and there is DB2 transaction log activity from multiple sources. When a lot of activity occurs from multiple sources, it is less likely that a single commit will have to wait for another commit (or the one-second time out). Do not adjust this parameter if you have a application with a single thread performing a series of commits (each commit could hit the one-second delay).
- **How to view or set:** To view the current value for a particular database follow these steps:
  1. Issue the DB2 command **get db cfg for xxxxxx** (where xxxxxx is the name of the application database) to list database configuration parameters.
  2. Look for "Group commit count (MINCOMMIT)".
  3. Set a new value by issuing the DB2 command **update db cfg for xxxxxx using mincommit n** (where n is a value between 1 and 25 inclusive).

  The new setting takes effect immediately.

  The following are several metrics that are related to DB2 MinCommit:

  1. The disk input/output wait can be observed on AIX with the command **vmstat 5**. This shows statistics every 5 seconds. Look for the wa column under the

CPU area.

2. The percentage of time a disk is active can be observed on AIX with the command **iostat 5**. This shows statistics every 5 seconds. Look for the %tm_act column.
3. The DB2 command **get snapshot for db on xxxxxx** (where xxxxxx is the name of the application database) shows counters for log pages read and log pages written.

- **Default value:**The default value is 1.
- **Recommended value:** MinCommit should be 1 or 2 (if the circumstance permits).

Session management

For additional information on setting session management parameters, see InfoCenter article about session programming model and environment.

WebSphere Application Server Enterprise Extensions Message Listener

WebSphere Application Server Enterprise Extensions provides support for Extended Messaging Support. This section contains tuning suggestions for the JMS Listener function which is part of Extended Messaging Support.

Maximum sessions

- **Short description:** This parameter represents the maximum number of concurrent message bean instances available to process a message at one time.
- **When to try adjusting:** If there are many messages to be processed and the message throughput (number of messages processed per minute) is very low, adjust this parameter for best throughput performance.
- **Actual benefit:** A 35% improvement in throughput was realized by adjusting the parameter from the default value to the recommended value. The workload/configuration information is as follows:
  - Message size = 10 bytes; persistent messages; 1 application server listening on 1 queue
  - System configuration = Netfinity 700 MHz; 4-way; 4GB RAM
- **How to view or set:** This parameter is set in the XML file jmsconfig.xml, which resides in the %WAS_HOME%\config directory. The current value is specified as follows:
  <MaxSessions> x <MaxSessions> (where x is the value to be used for number of sessions)
  Once this value is changed, the application server should be restarted to ensure the change was made.

  The utilization of this parameter can be seen by enabling the trace on cmm component.
- **Default value:** The default value for WebSphere Application Server Enterprise Extensions 4.0.1 is 5.
- **Recommended value:** The values range from 1 to 20.

  A value of 1 should be used if you want to serially process the messages, that is, only one message bean instance is used to process messages one after another.

  A value of 20 will give the best throughput. Increasing beyond this value will not increase the throughput. Based on the message type and the amount of work and the resources available, a value between 10 and 20 should be used for obtaining the maximum message throughput.

Multiple application servers listening on the same queue

- **Short description:** This consists of vertically cloned WebSphere Application Servers sharing queues of a single queue manager to increase the message throughput and as a failover mechanism.

  The increase in message throughput depends on various factors such as system resources and listener configuration. System resources refers to the number and power of the processors. Listener configuration is the number of sessions utilized and the JMS interactions. Included in the JMS interactions is the contention in sharing the access to the underlying MQ Server manager resources.
- **When to try adjusting:** Use this parameter in the following situations:
  1. When better message throughput is needed.
  2. When a fail-safe mode is required to retrieve the messages off a queue.
- **How to view or set:** This ranges from a single system with cloned servers to large-scale cloned WebSphere Application Server clustered with MQSeries queue managers.
- **Default value:** The default is a single application server listener on a single queue.
- **Recommended value:** If there are enough system resources available, message throughput can be increased using multiple application servers listening on a single queue. When systems have limited resources, a single server and listener give more message throughput.

Additional references

- WebSphere Application Server Development Best Practices for Performance and Scalability which describes development best practices for both Web applications containing servlets, JSP files, and JDBC connections, and enterprise applications containing enterprise bean components.
- iSeries performance documents including WebSphere Application Server 4.0 for iSeries Performance Considerations and links to the PTDV tool, Workload Estimator tool, and other documents.
- Redbook: WebSphere Application Server V3.5 Handbook (SG24-6161-00)
- Redbook: WebSphere Application Server V3 Performance Tuning Guide (SG24-5657-00)

Performance tool procedures

Starting Windows NT/2000 Performance Monitor

Follow these steps:
From the **Start** menu choose **Programs** > **Administrative Tools** > **Performance Monitor**