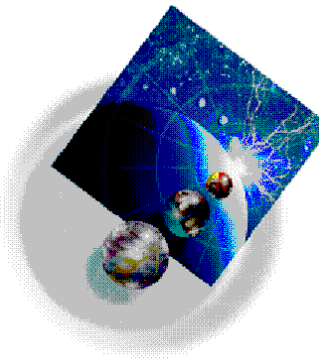
IBM WebSphere Application Server 3.0 Standard / Advanced Edition



Tuning Guide

IBM Software Solutions
4th Quarter 1999
IBM WebSphere Performance Team

In Search of the Magic Wand

So you want to tune the performance of your WebSphere Application Server installations? Where do you start? Where is that magic wand when you need it?

These are good questions; however, no hard, fast answers are available. At this point in time, the task ahead of you is more of a medieval art than an established science.

This document surveys the choices to consider. It outlines the various tuning points and recommends some initial starting values.

Three (3) main areas effect application server performance:

- The hardware capacity and settings
- The operating system settings
- The WebSphere Application Server product settings

Performance tuning is an ongoing learning experience. And of course, your results might vary from those presented in this paper. Check the performance Web site (http://water.raleigh.ibm.com) for periodic updates and additions to this information.

Please provide us feedback on tuning that you do to improve your application. Feedback may be sent to rewillen@us.ibm.com.

Hardware capacity and settings

This section discusses philosophies to consider when selecting and configuring the hardware on which your application servers will run.

Though it might sound trite, the simplest and best way to improve performance is to increase memory and processor speed.

From a performance perspective, each processor should be at least 300 MHz. Allow at least 512M memory per processor. See the JVM and Database sections for details on memory requirements.

Your Network Adapter card settings can also effect overall performance. Increasing the size of the Transmit and Receive queues will help handle network traffic, particularly for sites with significant static HTML content.

Operating System settings

This section discusses philosophies to consider when tuning the operating systems in your server environment.

Solaris (2.6-2.7)

You can modify Solaris settings to significantly improve WebSphere Application Server performance.

File descriptors (ulimit). You can adjust the number of file descriptors available. Set this parameter to at least 1024. If this parameter is too low, you will see a "*Too many files open*" error in the WebSphere Application Server stderr.log.
Check the man pages on ulimit for the syntax for different shells. For Korn shell (ksh) the command is:

ulimit -n 1024

Use *"ulimit -a"* to display the current values for all limitations on system resources.

The tcp_close_wait_interval parameter. This setting can significantly help performance on a heavily loaded web site. The default time wait interval is 2400000 ms. When high connection rates occur, a large backlog of TCP connections builds up and can slow the server down.

The performance team has seen visible delays of up to 4 minutes in which the server does not send any responses, but CPU utilization stays high, with all of the activity in system processes.

Set the tcp_close_wait_interval parameter as low as 30000 ms. As a starting point, use the following command to set it to 60000ms:

For Solaris 2.6:
ndd -get /dev/tcp tcp_close_wait_interval (to view the current setting)
*ndd -set /dev/tcp tcp_close_wait_interval 60000 (*to change the setting*)*

For Solaris 2.7:
ndd -get /dev/tcp tcp_time_wait_interval (to view the current setting)
*ndd -set /dev/tcp tcp_time_wait_interval 60000 (*to change the setting*)*

The performance team has heard of customer success stories from modifications to other Solaris tcp parameters such as tcp_conn_req_max_q, tcp_comm_hash_size, tcp_xmit_hiwat and others. We continue to run the product with the defaults for these parameters.

Although we have not seen significant performance differences due to raising these settings, your system might benefit. A good source of information about tuning the Solaris TCP/IP stack is available at:
http://www.rvs.uni-hannover.de/people/voeckler/tune/EN/tune.html

AIX (4.3.2/4.3.3)

File descriptors (ulimit). You can adjust the number of file descriptors available. For AIX, the default setting of 2000 is typically sufficient for most applications. If this parameter is too low, you will see "Memory allocation error". We have seen this when running 4 clones on an S80 24-way and needed to change the ulimit parameter to "unlimited". Check the man pages on ulimit for the syntax for different shells. For Korn shell (ksh) the command is:

ulimit -n 2000 (default)
ulimit -unlimited (large SMP machines with clones)

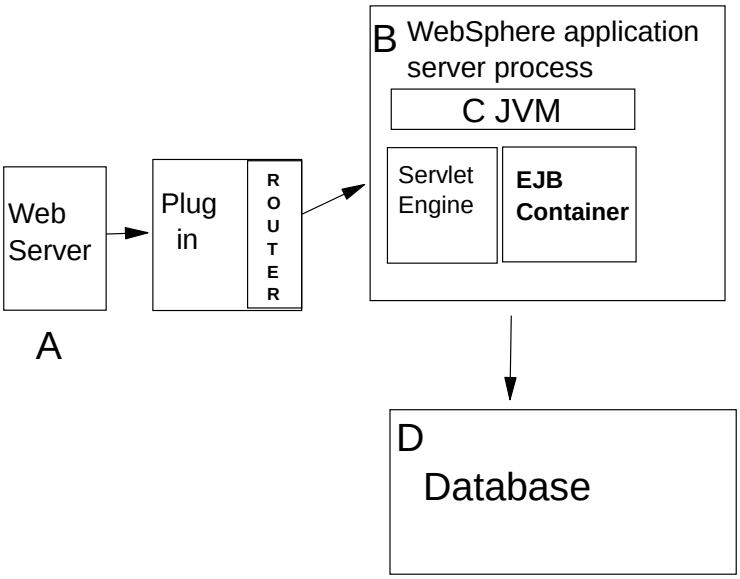
Use *"ulimit -a"* to display the current values for all limitations on system resources.

WebSphere Application Server settings

The WebSphere Application Server architecture provides several decision points that effect the overall performance of your applications. Because each decision point offers tradeoffs that you might view favorably or negatively depending on your circumstances, there is no magic, one-size-fits-all solution.

Each of the four (4) fundamental decision points has its own tuning options:

A. The Web server
B. The WebSphere application server process
C. The Java Virtual Machine (JVM)
D. The database



.

A. The Web server

The WebSphere Application Server product is designed to "plug-in" to several different Web server brands and versions. Each Web server-operating system combination features specific tuning parameters that effect your application performance.

This section discusses the performance tuning alternatives for the Web servers we have studied.

Web Servers on AIX

On AIX, the performance team has used the IBM HTTP Server (IHS) and the Netscape Enterprise Server in various tests. IHS is a multi-process, single-threaded server, whereas the Netscape server default configuration provides a single-process, multi-threaded server.

IBM HTTP Server (1.3.6) - AIX and Solaris

To modify IBM HTTP Server parameters, edit the server's active **httpd.conf** file. This section offers tuning guidance for a few of the parameters.

MaxClients. The value of the MaxClients parameter can significantly impact your application, particularly if it is too high. More is not always better! The optimum value depends on your application. In general:

* Use a lower MaxClients value for a simpler, CPU intensive application.
* Use a higher MaxClients value for a more complex, database intensive application with longer wait times
* Use a higher MaxClients value if your website contains a lot of static content

For example, exceptional performance on simple servlets like "HelloWorld" has been achieved using values as low as 25.

A good approach to tuning this parameter is to start with a setting of 50 and capture the performance under normal load. Repeat your tests with settings of 40 and 60. Use this data to refine your tuning. Use small increments and decrements rather than large ones.

During these tests, be sure to watch the server CPU utilization. Do not increase the MaxClient setting if the CPU utilization reaches 100% busy and doing so causes Server response time to exceed your response time criteria.

MaxClients - 50 (default *is 150*)

MinSpareServers, MaxSpareServers, and StartServers. This trio of settings can also impact your application performance. For optimum performance runs, keep the MaxSpareServers and

the StartServers parameters equal to reduce CPU expended creating and destroying httpd processes. This preallocates and keeps the specified number of processes so new processes will only be created and destroyed as the load approaches the specified number of processes (based on MinSpareServers)

MinSpareServers - 5 (default is 5)
MaxSpareServers - 50 (same as MaxClients - default is 10)
StartServers - 50 (same as MaxSpareServers)

More information on tuning the IBM HTTP server can be found at:
http://www.software.ibm.com/webservers/httpservers/doc/v136/misc/perf.html

Netscape Enterprise Server (3.6) - AIX and Solaris

Active threads. Use the "*Maximum number of simultaneous requests*" parameter in the Enterprise Server Manager to control the number of active threads within the Netscape Enterprise Server. This setting corresponds to the *RqThrottle* parameter in magnus.conf.

The default active thread limit is 512. Once the server reaches this limit, it will stop servicing new connections until it has finished with the old connections. If this setting is too low, your server might become throttled, resulting in degraded response times.

To tell if your Web Server is being throttled, consult its perfdump stats. Look at:

- The **WaitingThreads** count. If this number is getting close to 0 or is 0, then the server is not accepting new connections right now.

- The **BusyThreads** count. If the WaitingThreads count is close to 0 or is 0, then BusyThreads is probably very close to its limit.

- The **ActiveThreads** count. If this is close to its limit, the server is probably limiting itself.

Web servers on Windows NT

On Windows NT, the majority of our performance work has been done with Microsoft Internet Information Server (IIS). The performance team has done limited work with the Netscape Enterprise Server and the IBM HTTP server.

Microsoft Internet Information Server - Windows NT

Number of Expected Hits per Day. This parameter controls the memory IIS allocates for connections. Set this parameter in the Microsoft Management Console, Web Site Properties using the "Performance" tab.

Number of expected hits per day : Fewer than 10000 -----> More than 100000 (default is Fewer than 100000)

ListenBackLog parameter. If using IIS on Windows NT, you are likely to encounter failed connections under heavy load conditions (typically 100+ clients). This condition commonly results from IIS rejecting connections.

You can alleviate the condition by using the ListenBackLog parameter to raise the number of requests IIS keeps in its queue. If you intermittently experience "Netscape unable to locate server" from a Netscape browser when you are running heavy load, consider raising the parameter.

Use the operating system registry to set the ListenBackLog parameter. Issue "*regedit*" from a DOS Command line to access the registry. Locate the parameter under:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\ *ListenBackLog*

To modify the parameter, right-click it. Adjust the setting according to your server load. The default is 25, but the performance team has raised the ListenBackLog parameter as high as 200 without a negative impact on performance, and with an improvement in load handling.

IBM HTTP Server - Windows NT

ThreadsPerChild parameter. The *ThreadsPerChild* parameter in IHS on Windows NT is similar in function to the *MaxClients* parameter for IHS on AIX. The previous "IBM HTTP Server -- AIX" section described how to adjust MaxClients to improve performance.

Though the performance team has not performed significant IHS tuning on Windows NT, we suspect that you can derive tuning benefits by adjusting the ThreadsPerChild parameter in a similar way.

B. The WebSphere application server process

Each WebSphere applications server process has several parameters that will influence application performance.

Use the Topology tab of the WebSphere Application Server Administrative Console, shown in Figure 1 below, to configure and tune applications, servlet engines, EJB containers, application servers, and nodes in your administrative domain.

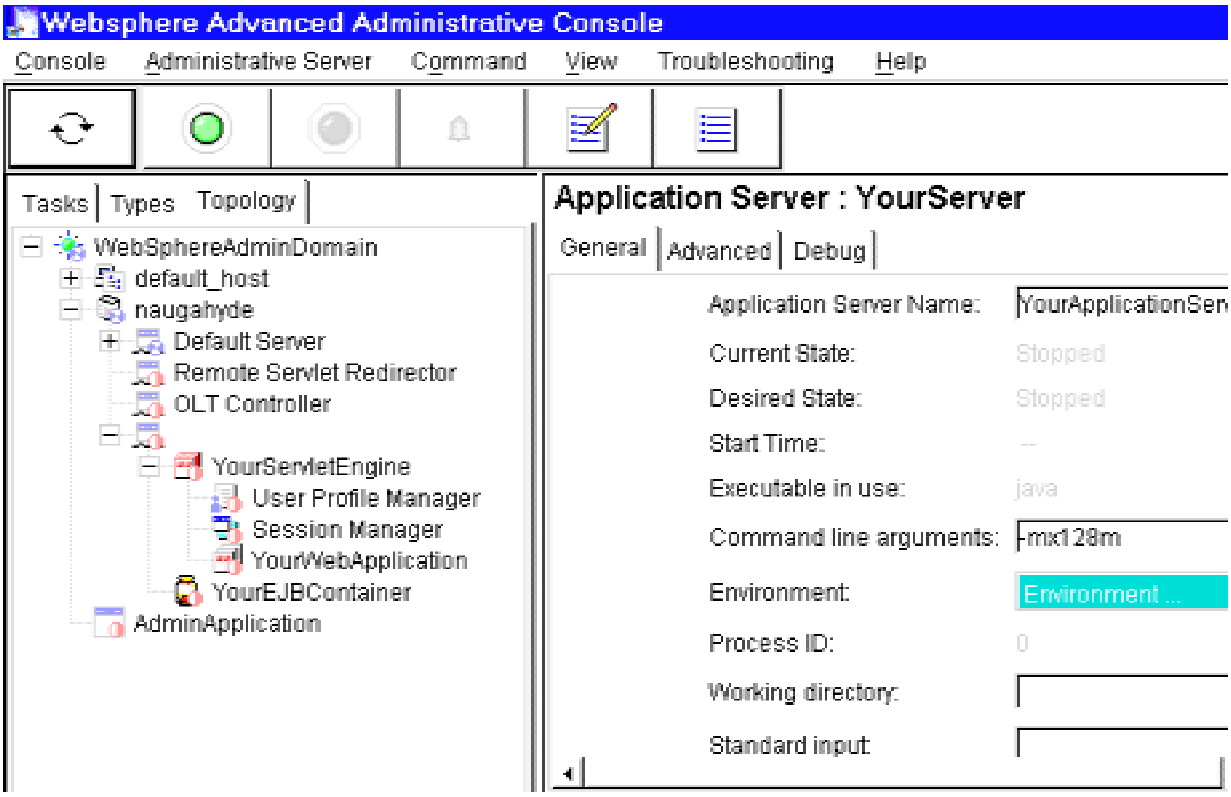


Figure 1: WebSphere Administrative Console: Application Server: Your Server

Application servers

The majority of the settings on the main Application Server panel shown in Figure 1 above will not significantly impact performance. The Command line arguments is one setting that will be common to use for both JVM and ORB parameters. The specific settings for the Command line arguments are described in the JVM and ORB sections below.

Servlet engines

Recall, each application server in your WebSphere Application Server product is comprised of an enterprise bean container and a servlet engine. To route servlet requests from the Web server to the servlet engines, the product establishes a transport queue between the Web server plug-in and each servlet engine.

Favor the OSE queue type. You can adjust the transport queue to improve performance. The queue type is of central importance. If distributing servlet requests from the Web server to servlets on a remote machine (known as "redirecting servlets"), you must use an IIOP-based

queue. The IIOP option is typically one to three times slower than the other option, Open Servlet Engine (OSE).
If not constrained to use IIOP, you should use OSE. Figure 2 below, shows where to configure this setting.

Servlet Engine: Advanced: Queue Type = OSE (default)

Fine-tune your OSE queue. After choosing OSE, you can perform additional tuning on the Settings Page of the servlet engine. This tuning is performed in the "Edit Servlet Engine Transport" panel shown at the bottom of Figure 2. This panel is selected by choosing the "Settings" button shown on the Servlet Engine: Your ServletEngine panel at the top of Figure 2.

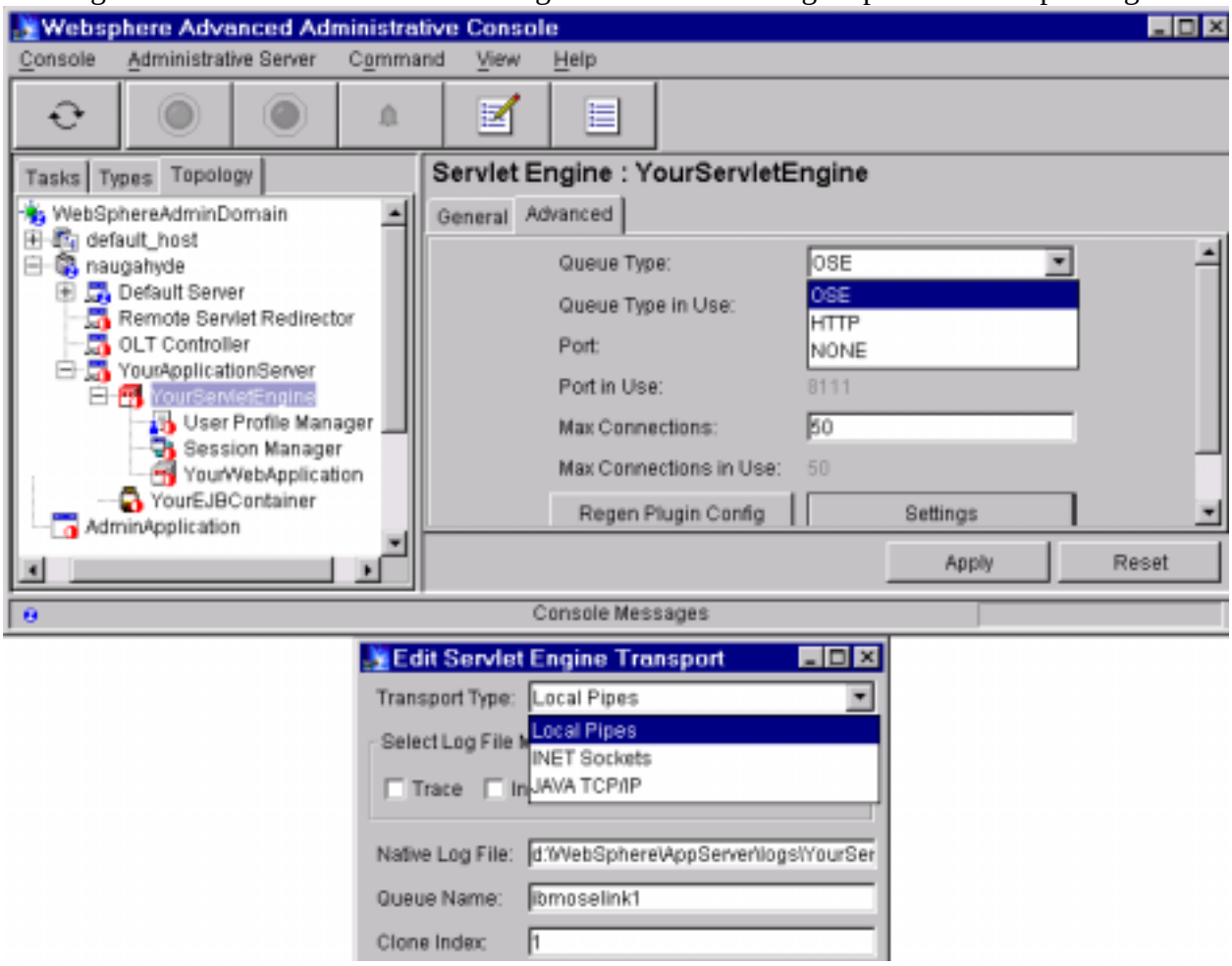


Figure 2: Configuring the Servlet Engine

For the transport type setting, you have three choices: Local Pipes, INET Sockets, and Java TCP/IP.

On AIX and Windows NT, "Local Pipes" is the default because it typically performs fastest. On Solaris, INET Sockets is the default and typically performs better under load than Local Pipes.

Servlet Engine: Advanced: Settings:Transport Type = Local Pipes (default on NT and AIX)

Servlet Engine: Advanced: Settings:Transport Type = INET Sockets (default on Solaris)

The transport type setting replaces the ose.outofproc.transport.type parameter from WebSphere Application Server Version 2.0.

Tune the maximum connections. Use the MaxConnections parameter to specify the number of connections to use for the communication channel between the Web server and a servlet engine. Each connection represents a request for a servlet.

Typically, you will achieve optimum performance and stability by setting the MaxConnections **less than or equal** to the number of threads or processes that are running within the Web server. In addition, the *maximum connection pool size* (see Database section) should be equal or lower than MaxConnections.

For example, on AIX if the IHS MaxClients parameter is set at 50, you would set both the MaxConnections and Maximum Connection pool size to 50 or less. For WebSphere Application Server 2.0 on AIX great results were achieved by setting the three parameters equal in ranges between 50-200. Early indications are that the optimum settings for WebSphere V3.0 are much lowe (typically less than 50).

Establishing the optimum relationship between the MaxClients, MaxConnections, and Maximum Connection Pool Size has provided some of the biggest customer performance improvements for Version 2, and is expected to be significant again for Version 3.

Servlet Engine: Advanced: MaxConnections = 50 (default)

This setting replaces ose.outofproc.link.cache.size from WebSphere 2.0.

Web Applications

With WebSphere Application Server Version 3, you can also set parameters specific to each Web Application you deploy. The settings can effect performance.

Use "auto reload" for development purposes only. WebSphere Application Server offers an "auto reload" capability. You can specify to automatically reload servlets in the Web application when their class files change.

The auto reload capability can simplify the testing and management of your Web site's applications by enabling you to quickly modify your site without restarting the WebSphere Application Server. However, this dynamic ability to reload servlets and associated polling effects performance negatively.

Once you are in production mode, you should turn off AutoReload.

WebApplication:your application:Advanced: AutoReload = false (default is true)

A Version 3 fixpack of the product will likely contain a Reload Interval Property, allowing you to control polling. Once the new property is available, consider leaving AutoReload "true" and establishing a very high interval for reload.

Note that if you want to use the WebSphere Resource Analyzer to monitor the performance of your servlets, you must explicitly configure the servlets in the WebSphere administrative domain. In other words, besides specifying the servlets' directories in the Web application classpath, you need to specify a name and other properties for each servlet. For more information, consult the help for the Resource Analyzer task on the Tasks tab of the WebSphere Administrative Console.

EJB Container

Cache Settings. To determine a rough approximation of the Cache absolute limit, multiply the number of beans active in any given transaction by the total number of concurrent transactions expected. Then add the number of active session bean instances. You can use the Resource Analyzer to view bean performance information.
EJB Container: your container:Cache Size = 2047 default
EJB Container: your container:Cache preferred limit = 2000 default
EJB Container: your container:Cache absolute limit = 2047 default
EJB Container: your container:Cache clean-up interval = 1000 default

Deployment Descriptors. When creating deployment descriptors for your EJBs, pay close attention to the beans function and define your descriptors accordingly. Performance benefits are realized by using the VA Java "read-only" attribute whenever appropriate.

Security

Turn off security when you do not need it. Security in WebSphere Application Server Version 3 is a global setting enabled through the WebSphere Administrative Console Task page. Enabling security has a significant impact on performance. In initial tests, the performance team saw degradation in the 50% range.

If you don't need security, preserve the default setting -- not enabled

Tasks:Security:Specify Global Settings:Enable Security: don't check

Note that the *Security enabled* property in the application server settings does not work. Please make sure you use the Global Setting task on the Tasks tab to enable security. You will be enabling it for all application servers.

Fine-tune the security cache timeout for your environment. If you do enable WebSphere Application Server security, the security cache timeout can influence performance. The timeout specifies how often to refresh the security-related caches.

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

In a simple 20 minute performance run, the performance team raised the cache timeout from 600 seconds to 6000 seconds so that it did not timeout during the run, and achieved a 40% performance improvement. You can experiment to find the best trade-off for your application based on your site usage patterns.

Tasks:Security:Specify Global Settings:Security Cache Timeout: 6000

Configure SSL sessions appropriately. The SSLV3Timeout value is the time interval after which SSL sessions are renegotiated. This is a high setting, and probably won't have any significant impact from modification. By default, it is set to 9600 seconds.

You can modify SSLV3Timeout and other Secure Association Service (SAS) properties by editing "*sas.server.props*" and "*sas.client.props*" files. The files reside in the <applicationserver_root>\properties directory, where <applicationserver_root> is the directory in which you installed WebSphere Application Server.

Also note that the SAS feature establishes an SSL connection only if it goes out of the ORB (to another ORB). Therefore, if all the beans are CO-located within an ORB, then the SSL used by SAS should not hinder performance.

Object Request Brokers (ORBs)

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

Use the Command Line property of the "Default Server" or any additional WebSphere application server you configure in the WebSphere administrative domain to set these parameters:

Deploying EJBs in the same process. Normally RMI is Pass By Value (the callee gets a copy of caller's args), regardless of in- or out-of-process. For in-process calls (ie., one EJB to another), local copies can be disabled allowing the callee to get a reference to caller's args. In WebSphere 3.0 GA, the following two parameters both need to be added to the Command Line property to disable local copies:

-Djavax.rmi.CORBA.UtilClass=com.ibm.CORBA.iiop.Util
-Dcom.ibm.CORBA.iiop.noLocalCopies=true

In a future release, a GUI parameter will be added for this setting.

Location Service (LSD)

Disabling the LocationService is a dangerous operation. It prevents failover for entity beans and may break transaction recovery. Though disabling LSD will boost performance, it is not recommended.

If you do need to disable the Location Service, use the Command Line property of the "Default Server" or any additional WebSphere application server you configure in the WebSphere administrative domain to set this parameter.

-disableLocationService

C. Java Virtual Machines (JVMs)

Selecting a JVM

WebSphere Application Server requires a JVM. Though WebSphere ships a default JVM, you can use alternate JVMs, as long as they are supported. (See the software requirements sections of the WebSphere Application Server Getting Started book for details).

On Windows NT, several vendors offer JVMs. The IBM Developer Kit and Runtime Environment for Windows, Java Technology Edition, Version 1.1.7p that ships with WebSphere Application Server Version 3 is the supported JVM and typically the best performing Windows NT JVM for WebSphere applications.

On Solaris, SunSoft offers a JVM tuned specifically for Solaris. It will typically outperform the base Solaris reference implementation from JavaSoft. WebSphere Application Server Version 3 ships the JavaSoft reference implementation.

Tuning the JVM

The JVM offers several tuning parameters that will impact the performance of WebSphere application servers (which are primarily Java applications), as well as the performance of your own applications. In WebSphere Application Server Version 3, you can set JVM parameters in the Command Line property of the "Default Server" or any additional WebSphere application server you configure in the WebSphere administrative domain.

JIT and Garbage Collection. The JIT and garbage collection settings significantly effect performance. In all cases, you will want to with JIT on, which is the default. In most cases you will also want asynchronous garbage collection on and class garbage collection on, again these are the defaults.

On AIX, forcing garbage collection prior to the mandatory (out-of-memoy) garbage collection may improve performance. With a simple Servlet-EJB Entity Bean (Bean Managed) application, a 15% performance (requests/sec) improvement was seen by forcing a garbage collection (i.e.

System.gc()) before the mandatory garbage collection was invoked. The mandatory garbage collection normally occurred with this application after approximately 2000 hits. The Servlet was modified to force a garbage collection after 1900 hits. The Default Server was started with -ms and -mx values of 128m.

The performance team has seen cases on Solaris using enterprise beans where "-noasyncgc" can actually improve performance. You can also turn off class garbage collection to enable 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 WebSphere application server you configure in the WebSphere administrative domain to set the JVM parameters:

-noasyncgc (using -noasyncgc on Solaris may occassionally improve performance)
-noclassgc ("no" will enable more class reuse and occassionally improve performance)

Heap Size Settings. Java -mx and -ms are used to set the maximum and the minimum (starting) heap size for the JVM. In general, increasing the size of the java heap improves throughput up until the point where the heap no longer resides in physical memory. Once, the heap begins swapping to disk Java performance drastically suffers. Therefore, the -mx heap setting should be set small enough to contain the heap within physical memory. Note that the physical memory usage must be shared between the JVM and the other applications, for instance, the database. It is better to err on the side of safety and use a smaller heap (say 64MB on smaller memory machines).

If there is enough memory on a system, then a maximum heap of 128MB on a smaller machine (< 1GB physical memory), and 256MB (>1GB physical memory) on a larger machine may be appropriate.

If performance runs are being conducted and highly repeatable results are needed then the -ms and -mx values should be set to the same value. This will eliminate any heap growth during the run. For production system's where the working set size of the java applications are not well understood a -ms setting of 1/4 the -mx setting is a good starting place. The JVM will then try to adapt the size of the heap to the working set of the Java application.

-mx = 128m (default is 128m; consider using 256m for >4 way machines)
*-ms = 32m (*1/4 mx for production; for performance runs set ms to be equal to mx)

Stack Size Settings. Use the oss and ss parameters to set java and native thread size stacks within the JVM. For WebSphere Application Server 3.0, performance test runs are using the default settings.

D. The Database

WebSphere Application Server Version 3 is tightly integrated with a supported database of your choice. (See the Getting Started book software requirements for details). 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 your application.

If your application uses WebSphere Session State, JDBC Database Connection Pooling or enterprise beans, pay special attention to how you configure these resources and their database settings within the WebSphere administrative domain. During WebSphere Application Server installation, when using DB2 a database named "WAS" is typically established, although you can specify a different name. This document assumes you used "WAS."

Database Access

Most Java applications use JDBC to access databases. In any access to a database, the initial database connection is a very expensive operation. WebSphere Application Server V3 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 your application as well as for enterprise beans using the database. Using the connection pooling capabilities, you can realize performance improvements of approximately 20x.

WebSphere Data Source Settings

The Database connection pool options are specified through the Data Source object. The connection pool size will effect the performance of your application. Better performance is generally achieved if the connection pool size is set lower than the MaxConnections in the Servlet Engine. In our tests so far, the performance team has often found that lower settings (10-30 connections) perform better than higher (100+) settings. For example, a 100 client load run against the Connection Pool Test servlet on our NT system performs best with a maximum connection pool size of 5. The performance with different settings is as follows:

Max connection pool size	2	5	10	50
Transactions per second	181	203	186	137

On UNIX platforms, a separate DB2 process is created for each connection. This will quickly impact performance on low memory systems and errors can occur. If you are running with 512M or less, it is recommended you use 10 or fewer connections.

Data Source:YourDataSource:Minimum connection pool size: 1 (default)
Data Source:YourDataSource:Maximum connection pool size: 10 (default is 30)
Data Source:YourDataSource:Connection timeout: 300 (default)
Data Source:YourDataSource:Idle timeout: 1800 (default)
Data Source:YourDataSource:Orphan timeout: 1800 (default)

The Resource Analyzer can be used to view the Percentage of Database Connections in use. Use this value to adjust the size of the connection pool and then iteratively tune this setting in conjunction with the MaxConnections setting in the Servlet Engine.

In our EJB primitive tests on NT, we looked at the relationship between the number of concurrent user requests, connection pool size, and prepared statement cache size. Better performance occurred when the:

1. Number of concurrent user threads <= Maximum connection pool size
2. PrepStmtCache > diffPrepStmts * min(concurrent user threads, connectionPoolSize)

For example, for 20 concurrent user threads, you would set Maximum connection pool size to at least 20. If each client is just doing an entity read, and there are two different prepStmts, load and save, the prepared statement cache size should be at least 2*20=40.

Set PrepStmtCacheSize through the server command line:

-Dcom.ibm.ejs.dbm.PrepStmtCacheSize=40

The Connection Pool settings are directly related to the number of connections that your database server is configured to support. If you raise the maximum number of connections in the pool, and do not raise the corresponding settings in the database, your application will fail. You will get SQL exception errors in the stderr.log. The specific parameters on the Database are described in the Database section below.

Session Management

Keep sessions in memory whenever possible. If your application requires sessions to be maintained across requests, WebSphere Application Server supports the Servlet API HttpSession interface. In WebSphere V3, sessions can be maintained in memory or stored persistently in a database via JDBC or enterprise beans.

- Storing sessions in memory typically will be much faster than using a persistent backing store; however, there are significant scalability and failover limitations with the in-memory storage.
- Using a database will allow session to be easily shared in clustered and cloned environments.

Our internal tests currently show application performance degradation of approximately 50% when sessions are stored in a database, but this will vary depending on the application.

Close finished sessions promptly. In developing applications that use sessions, performance will benefit if your application can detect the end of a session and explicitly close the session. This is particularly important if sessions are being stored in memory. If sessions are not released, they will be closed by WebSphere Application Server based on the Invalidate Session timeout setting. The default is 1800 seconds (30 minutes). If your application cannot detect when a client leaves the site and has to rely on session timeouts, the Invalidate Session setting is extremely important to monitor. If traffic is heavy and a lot of sessions are being opened, but not

closed, performance can degrade significantly. On simple tests, we witnessed approximately 50% performance differential when sessions were explicitly closed versus having them remain open for 30 minutes. For active web sites, significant performance improvements have been seen by lowering the session time-out. Use the Resource Analyzer to see how many sessions are open.

Servlet Engine:Session Manager:Intervals:Invalidate Time: 600 seconds (default 1800)

Configure the Session Manager database connections. If you choose to store session in a backend database, the WebSphere Application Server Session Manager has a parameter to specify the number of connections for the session pool. In WebSphere Application Server 3.0 GA, the pool is separate from the JDBC Data Source specifications (this changes with 3.0.2). Our internal tests show significant performance degradation can occur when the number of connections is less than the number of concurrent users your application is processing.

Servlet Engine:Session Manager:Persistence:Number of Connections: number of concurrent users (default is 30)

Tune the Session Manager. The Session Manager also has a set of tuning options. The performance team has not done much significant work with these yet. We have seen performance improvements of 15%-30% in simple persistent session workloads from Using Cache = Yes.

SessionMgr:Session Manager:Tuning:Using Cache=Yes (default is No)

The remaining settings and their defaults are below.

SessionMgr:Session Manager:Tuning:Using Multirow Sessions=No
SessionMgr:Session Manager:Tuning:Using Manual Update=No
SessionMgr:Session Manager:Tuning:Using Native Access=No (N/A for V3)
SessionMgr:Session Manager:Tuning:Using Overflow=Yes
SessionMgr:Session Manager:Tuning:Base Memory Size=1000

The Session Manager Number of Connections parameter is directly related to the number of connections that your database server is configured to support. If you raise the number of connections for sessions, and do not raise the corresponding settings in the database, your application will fail. The specific parameters on the Database are described in the Database section below.

DB2

Database Location

The DB2 database can reside on the same server as WebSphere Application Server or on a remote machine. For scalability, it is likely you established the database on a separate machine, particularly if you are using clustering.

If you haven't put your database on a separate machine, consider doing so. Depending on your machine utilization, significant improvements can be obtained by offloading the database to a separate machine.

On UNIX platforms, whether or not the DB2 server resides on a machine with WebSphere Application Server, you should configure your DB2 Application Databases to use TCP sockets for database communication. The directions for configuring DB2 on UNIX can be found in the WebSphere Application Server Getting Started documentation and include specifics for setting DB2COMM for TCP/IP and corresponding changes required in the etc/service file. The WebSphere administrative repository, typically a DB2 database named WAS, can use the default local configuration for its communication.

Choice of JDBC driver for DB2

Your application can use type 2, 3, and 4 JDBC drivers. The DB2 Type 2 application driver typically performs better than Types 3 or 4.

DB2 Settings

DB2 has many parameters you can configure to optimize database performance. We will not attempt to make this document an all encompassing DB2 tuning document. A few required settings are listed below, but for complete DB2 tuning information, please refer to the DB2 System Monitor Guide and Reference.

The WAS database is used by WebSphere Application Server for configuration information. Raise the Application Heap Size for this database to at least 256M.

Application Heap Size = 256M (default 128M)

The Application databases that your application connects to through WebSphere application server might also need setting adjustments. When you configure the Data Source settings for the databases, ensure that the DB2 MaxAppls setting is greater than the Maximum number of connections for your 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 minimum setting for Buffpage has to be double the MaxAppls setting.

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 mgr) x # of clones (default is 40)

Buffpage = 2 x MaxAppls (minimum setting)

After you have calculated the MaxAppls settings for the WAS database and each of your 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

Oracle

Choice of JDBC driver for Oracle

For Oracle, only Type 4 is officially supported at this time. The Oracle Type 2 driver will not work with the IBM NT JVM.

Tuning Parameter Summary

The tuning guide is just a STARTING POINT. Optimum settings will vary significantly depending on your application, user load, and environment. Please do not just blindly change settings!

Parameters flagged in the Key Tuning Parameter column have been seen to make a significant APPLICATION specific difference in performance. Because these are APPLICATION dependent the appropriate setting for your application and environment must be determined through iterative tuning and monitoring using the Resource Analyzer. Other parameters are either generally applicable to all applications or are expected to yield less significant performance improvements through tuning. Your mileage will vary. As always, we welcome feedback on your tuning successes and failures.

parameter	Default	Key Tuning Parm	Comments
Operating System Settings			
File Descriptors ulimit -n	Solaris - to low AIX - 2000	Yes (Solaris)	Usually requires raising on Solaris
tcp_close_wait_interval	240000ms	Yes	Often requires tuning on Solaris, significant performance impact
HTTP server - AIX IHS 1.3.3 (http.conf)			
MaxClients	150	Yes	Often requires tuning (lower often better). Significant performance impact
MinSpareServers	5		
MaxSpareServers	10		
Start servers	5		
WebSphere 3.0 Admin Console)			
Servlet Engine Parameters			
Servlet Engine: Advanced: Queue Type	OSE	Yes	Use OSE
Servlet Engine: Advanced: Transport Type	Local pipes	Yes	Use INET Sockets on Solaris. Use Local

Settings			Pipes on Window NT and AIX
Servlet Engine: Advanced: MaxConnections Settings	50	Yes	Often requires tuning. Tune in conjunction with MaxClients and Maximum Database Connections. Lower often better.
Web Application Parameters			
WebApplication: Advanced: AutoReload	true		Recommend false
Java Parameters			
Default Server: Command Line: -compiler	blank (-compiler)		Make sure that the JIT is enabled
Default Server: Command Line: -classgc	blank (-classgc)		This parameter typically only makes minor difference one way or another
Default Server: Command Line: -asyncgc	blank (-asyncgc)		
Default Server: Command Line: -mx	128M	Yes	Set this parameter based on your system memory and app. Too big a heap can cause long GC pauses.
Default Server: Command Line: -ms	None (JVM specific default)		
Default Server: Command Line: -oss			
Default Server: Command Line: -ss			
Database Access			
Data Source: YourDataSource: Mimimum connection pool size	1		
Data Source:	30	Yes	Often requires tuning.

YourDataSource: Maximum connection pool size			Tune in conjunction with MaxClients and MaxConnections. Lower often better.
Data Source: YourDataSource: Connection timeout	300		
Data Source: YourDataSource: Idle timeout	1,800		
Data Source: YourDataSource: Orphan timeout	1,800		
-Dcom.ibm.ejs.dbm.PrepStmtCacheSize	40		
Session Manager			
Session Manager: Persistence: Number of Connections	30		
Session Manager: Persistence: Intervals: Invalidate Time	1800 seconds		
Session Manager: Tuning: Using Multirow Sessions	No		
Session Manager: Tuning: Using Cache	No		
Session Manager: Tuning: Using Manual Update	No		
Session Manager: Tuning: Using Overflow	Yes		
Session Manager: Tuning: Base Memory Size	1,000		
DB2			
Application Heap	128M	Yes	256M required

Size			
Maxappls	40		May need to raise for application to run. Tune in conjunction with Maximum Database Connections.
MaxAgents	40		May need to raise for application to run. Tune in conjunction with Maximum Database Connections
Bufpool			
DB2 Communication	Shared memory		TCP Sockets for UNIX