



BEA WebLogic Server™

BEA WebLogic Server Performance and Tuning

Version 8.1
Revised: June 28, 2006

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

Audience	xiv
eDocs Web Site	xiv
How to Print the Document	xiv
Related Information	xiv
Contact Us!	xv
Documentation Conventions	xv

1. Top Tuning Recommendations for WebLogic Server

Understand Your Performance Objectives	1-2
Tune the Operating System	1-2
UNIX Tuning Parameters	1-2
Solaris TCP Tuning Parameters	1-2
HP-UX Tuning Parameters	1-3
AIX Tuning Parameters	1-3
Linux Tuning Parameters	1-3
Windows Tuning Parameters	1-4
Optimize Your Database	1-4
General Suggestions	1-4
Database-Specific Tuning	1-4
Oracle	1-5
Microsoft SQL Server	1-6

Sybase	1-6
Identify the Best JVM Settings	1-7
Sun JDK	1-7
JRockit JDK	1-7
Tune WebLogic Server Performance Parameters	1-7
Monitor Disk and CPU Utilization	1-8
Monitor Data Transfers Across the Network	1-9
Check For Frequent Standard I/O or Logging	1-10
Locate Bottlenecks in Your Applications	1-10
Tune Your Application	1-10
EJBs	1-10
JSPs and Servlets	1-11
JMS	1-11
JDBC	1-11

2. Tuning Hardware, Operating System, and Network Performance

Hardware Tuning	2-1
Supported Platforms	2-2
Operating System Tuning	2-3
Solaris Tuning Parameters	2-3
Setting TCP Parameters With the ndd Command	2-4
Setting Parameters In the /etc/system File	2-5
CE Gigabit Network Card Settings	2-5
Linux Tuning Parameters	2-6
HP-UX Tuning Parameters	2-7
Other Operating System Tuning Information	2-7
Network Performance	2-8
Determining Network Bandwidth	2-8

LAN Infrastructure	2-9
------------------------------	-----

3. Tuning Java Virtual Machines (JVMs)

JVM Tuning Considerations	3-2
Which JVM for Your System?	3-3
Changing To a Different JVM	3-3
JVM Heap Size and Garbage Collection	3-3
Choosing a Garbage Collection Scheme	3-4
Using Verbose Garbage Collection to Determine Heap Size	3-5
Specifying Heap Size Values	3-6
Using WebLogic Startup Scripts to Set Heap Size	3-7
Java Heap Size Options	3-8
Automatically Logging Low Memory Conditions	3-9
Manually Requesting Garbage Collection	3-10
Setting Java HotSpot VM Options	3-10
Standard HotSpot VM Options for Windows, Solaris, and Linux	3-11
Non-Standard HotSpot VM Options for Windows, Solaris, and Linux	3-12

4. Tuning WebLogic Server

Setting Java Parameters for Starting WebLogic Server	4-1
Setting Performance-Related Configuration Parameters	4-2
Development vs. Production Mode Default Tuning Values	4-4
Using WebLogic Server “Native IO” Performance Packs	4-5
Which Platforms Have Performance Packs?	4-5
Enabling Performance Packs	4-5
Tuning the Default Execute Queue Threads	4-6
Should You Modify the Default Thread Count?	4-7
Scenarios for Modifying the Default Thread Count	4-7

Modifying the Default Thread Count	4-9
Assigning Applications to Execute Queues	4-9
Allocating Execute Threads to Act as Socket Readers	4-9
Setting the Number of Socket Reader Threads For a Server Instance	4-10
Setting the Number of Socket Reader Threads on Client Machines	4-10
Tuning Execute Queues for Overflow Conditions	4-10
Tuning the Execute Thread Detection Behavior	4-12
Tuning Connection Backlog Buffering	4-14
How JDBC Connection Pools Enhance Performance	4-14
Tuning JDBC Connection Pool Initial Capacity	4-15
Tuning JDBC Connection Pool Maximum Capacity	4-15
Caching Prepared and Callable Statements	4-16
Setting Your Java Compiler	4-16
Changing Compilers in the Administration Console	4-16
Setting Your Compiler in weblogic.xml	4-17
Compiling EJB Container Classes	4-17
Compiling on UNIX	4-17
Using WebLogic Server Clusters to Improve Performance	4-18
Scalability and High Availability	4-18
How to Ensure Scalability for WebLogic Clusters	4-19
Database Bottlenecks	4-19
Session Replication	4-19
Invalidation of Entity EJBs	4-20
Invalidation of HTTP sessions	4-20
JNDI Binding, Unbinding and Rebinding	4-20
Performance Considerations When Running Multiple Server Instances on Multi-CPU Machines	4-20
Monitoring a WebLogic Server Domain	4-21

5. Tuning WebLogic Server EJBs

Setting Performance-Related weblogic-ejb-jar.xml Parameters	5-1
Setting EJB Pool Size for Session and Message-Driven Beans	5-2
Using a Free Pool to Improve Stateless Session Bean Performance	5-3
Allocating Pool Size for Entity Beans	5-4
Tuning Pool Size for Stateless Sessions Beans at Startup	5-4
Setting Caching Size for Stateful Session and Entity Beans	5-4
Activation and Passivation of Stateful Session EJBs	5-5
Deferring Database Locking.	5-5
Setting Transaction Isolation Level	5-6
Setting Performance-Related weblogic-cmp-jar.xml Parameters	5-6
Tuning In Response to Monitoring Statistics	5-7
Cache Miss Ratio	5-7
Lock Waiter Ratio.	5-8
Lock Timeout Ratio	5-8
Pool Miss Ratio	5-9
Destroyed Bean Ratio.	5-9
Pool Timeout Ratio.	5-10
Transaction Rollback Ratio	5-10
Transaction Timeout Ratio	5-11
Other Performance Improvement Strategies	5-11
Application-Level Caching	5-11
Batch Operations	5-12
Distributing Transactions Across EJBs in a WebLogic Server Cluster	5-12
Indexing with a Version Column	5-12

6. Tuning WebLogic Server Applications

Using Performance Analysis Tools	6-2
--	-----

Using the JProbe Profiler	6-2
Using the Optimizeit Profiler	6-2
JDBC Application Tuning	6-2
JMS Application Tuning	6-3
EJB Application Tuning	6-3
Web Services Tuning	6-3
Managing Sessions	6-4
Managing Session Persistence	6-4
Minimizing Sessions	6-5
Using Execute Queues to Control Thread Usage	6-5
Creating Execute Queues	6-6
Assigning Servlets and JSPs to Execute Queues	6-8
Assigning EJBs and RMI Objects to Execute Queues	6-9

A. Related Reading: Performance Tools and Information

BEA Systems, Inc. Information	A-2
Sun Microsystems Information	A-2
Linux OS Information	A-3
Hewlett-Packard Company Information	A-4
Microsoft Information	A-4
Web Performance Tuning Information	A-5
Network Performance Tools	A-6
Load Testing Tools	A-6
Performance Analysis Tools	A-6
Production Performance Management	A-7
Benchmarking Information	A-7
Java Virtual Machine (JVM) Information	A-8
Enterprise JavaBeans Information	A-9

Java Message Service (JMS) InformationA-9

Java Database Connectivity (JDBC) InformationA-10

General Performance InformationA-10

About This Document

To achieve the best performance for your WebLogic Server™ platform, you need to optimize the performance of the components that constitute the WebLogic Server environment. This document provides the following performance-related information:

- [Chapter 1, “Top Tuning Recommendations for WebLogic Server,”](#) discusses the most frequently recommended steps for achieving optimal performance tuning for applications running on WebLogic Server.
- [Chapter 2, “Tuning Hardware, Operating System, and Network Performance,”](#) discusses hardware, operating system, and network performance issues.
- [Chapter 3, “Tuning Java Virtual Machines \(JVMs\),”](#) discusses JVM tuning considerations.
- [Chapter 4, “Tuning WebLogic Server,”](#) contains information on how to tune WebLogic Server to match your application needs.
- [Chapter 5, “Tuning WebLogic Server EJBs,”](#) describes how to tune WebLogic Server Enterprise Java Beans to match your application needs.
- [Chapter 6, “Tuning WebLogic Server Applications,”](#) discusses application tuning considerations.
- [Appendix A, “Related Reading: Performance Tools and Information,”](#) provides an extensive performance-related reading list.

The document also contains an index.

Audience

This document is written for people who monitor performance and tune the components in a WebLogic Server platform. It is assumed that readers know server administration and hardware performance tuning fundamentals, the WebLogic Server platform, XML, and the Java programming language.

eDocs Web Site

BEA product documentation is available on the [BEA corporate Web site](http://www.bea.com) at <http://www.bea.com>. From the BEA Home page, click on Product Documentation and follow the link to BEA WebLogic Server Documents. Or you can go directly to the WebLogic Server Product Documentation page at <http://edocs.bea.com/wls/docs81/index.html>.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the [Adobe Web site](http://www.adobe.com) at <http://www.adobe.com>.

Related Information

For related information about administering and tuning WebLogic Server, see:

- *WebLogic Server Capacity Planning Guide* at <http://edocs.bea.com/wls/docs81/pdf/cappplanpublic.pdf>.
- *Configuring and Managing WebLogic Server* at <http://edocs.bea.com/wls/docs81/adminguide/index.html>.
- BEA [dev2dev](#) Web site.
- The WebLogic Server performance “weblogic.developer.interest.performance” newsgroup available on the [BEA Newsgroup](#) server.

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, and the title and document date of your documentation. If you have questions about this version of BEA WebLogic Server, or if you have problems installing and running it, contact BEA Customer Support through BEA WebSupport at <http://www.bea.com>, or by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Usage
Ctrl+Tab	Keys you press simultaneously.
<i>italics</i>	Emphasis and book titles.

Convention	Usage
monospace text	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard. <i>Examples:</i> import java.util.Enumeration; chmod u+w * config/examples/applications .java config.xml float
<i>monospace italic text</i>	Variables in code. <i>Example:</i> String <i>CustomerName</i> ;
UPPERCASE TEXT	Device names, environment variables, and logical operators. <i>Examples:</i> LPT1 BEA_HOME OR
{ }	A set of choices in a syntax line.
[]	Optional items in a syntax line. <i>Example:</i> java utils.MulticastTest -n <i>name</i> -a <i>address</i> [-p <i>portnumber</i>] [-t <i>timeout</i>] [-s <i>send</i>]
	Separates mutually exclusive choices in a syntax line. <i>Example:</i> java weblogic.deploy [list deploy undeploy update] password {application} {source}

Convention	Usage
. . .	<p>Indicates one of the following in a command line:</p> <ul style="list-style-type: none"> • An argument can be repeated several times in the command line. • The statement omits additional optional arguments. • You can enter additional parameters, values, or other information
.	Indicates the omission of items from a code example or from a syntax line.

About This Document

Top Tuning Recommendations for WebLogic Server

Performance tuning WebLogic Server and your WebLogic Server application is a complex and iterative process. To get you started, we have created a “top ten” list of recommendations to help you optimize your application’s performance. These tuning techniques are applicable to nearly all WebLogic applications. Although we highly recommend performing these tasks in the sequence they are presented, this isn’t a requirement.

- [“Understand Your Performance Objectives” on page 1-2](#)
- [“Tune the Operating System” on page 1-2](#)
- [“Optimize Your Database” on page 1-4](#)
- [“Identify the Best JVM Settings” on page 1-7](#)
- [“Tune WebLogic Server Performance Parameters” on page 1-7](#)
- [“Monitor Disk and CPU Utilization” on page 1-8](#)
- [“Monitor Data Transfers Across the Network” on page 1-9](#)
- [“Check For Frequent Standard I/O or Logging” on page 1-10](#)
- [“Locate Bottlenecks in Your Applications” on page 1-10](#)
- [“Tune Your Application” on page 1-10](#)

Understand Your Performance Objectives

Gather information about the level of activity expected on your server, the anticipated number of users, the number of requests, acceptable response time, and an optimal hardware configuration (e.g., fast CPU, disk size vs. speed, sufficient memory, and so on.).

There is no single formula for determining your hardware requirements. The process of determining what type of hardware and software configuration is required to meet application needs adequately is called capacity planning. Capacity planning requires assessment of your system performance goals and an understanding of your application. Capacity planning for server hardware should focus on maximum performance requirements.

For more information about capacity planning for WebLogic Server, see the [BEA WebLogic Server Capacity Planning Guide](#).

Tune the Operating System

Each operating system sets default tuning parameters differently. For Windows platforms, the default settings are usually sufficient. However, the UNIX and Linux operating systems usually need to be tuned appropriately.

UNIX Tuning Parameters

Use the following guidelines when tuning UNIX operating systems supported by WebLogic Server.

Solaris TCP Tuning Parameters

For better TCP (transmission control protocol) socket performance, set the `tcp_time_wait_interval` parameter as follows:

```
ndd -set /dev/tcp tcp_time_wait_interval 60000
```

This parameter determines the time interval that a TCP socket is kept alive after issuing a close call. The default value of this parameter on Solaris is four minutes. When a large number of clients connect for a short amount of time, holding these socket resources can have a significant negative impact on performance. Setting this parameter to a value of 60000 (60 seconds) has shown a significant throughput enhancement when running benchmark JSP tests on Solaris. You might want to reduce this setting further if the server gets backed up with a queue of half-opened connections.

Note: Prior to Solaris 2.7, the `tcp_time_wait_interval` parameter was called `tcp_close_wait_interval`.

For additional recommended Solaris tunable settings, see:

- [“Setting TCP Parameters With the `ndd` Command” on page 2-4](#)
- [“Setting Parameters In the `/etc/system` File” on page 2-5](#)
- [“CE Gigabit Network Card Settings” on page 2-5](#)

For more information about Solaris tuning options, see:

- [Solaris Tunable Parameters Reference Manual](#) (Solaris 8)
- [Solaris Tunable Parameters Reference Manual](#) (Solaris 9)

HP-UX Tuning Parameters

For HP-UX tuning information, see:

- [Tunable Kernel Parameters](#) reference documentation.
- [Java Performance Tuning on HP-UX](#)

AIX Tuning Parameters

See the [AIX 5L Version 5.2 Performance Management Guide](#).

Linux Tuning Parameters

For better packet transfer performance, set the `/sbin/ifconfig lo mtu` parameter as follows:

```
/sbin/ifconfig lo mtu 1500
```

The `mtu` (maximum transfer unit) parameter refers to largest number of bytes that a packet can carry over the network. If the packet size is set too low, then your network performance will decrease due to fragmented data.

For additional recommended Linux tunable settings for WebLogic Server, see [“Linux Tuning Parameters” on page 2-6](#). For general information about Linux tuning, consult your Linux vendor’s documentation. Also, the [Ipsysctl Tutorial 1.0.4](#) describes all of the IP options provided by Linux.

Windows Tuning Parameters

For Windows platforms, the default settings are usually sufficient. For more information about Windows 2000 tuning options, see:

- The [Microsoft Windows 2000 TCP/IP Implementation Details](#) white paper.
- The [Windows 2000 Performance Tuning](#) white paper.

Optimize Your Database

Your database can be a major enterprise-level bottleneck. Configure your database for optimal performance by following the tuning guidelines in this section and in the product documentation for the database you are using.

General Suggestions

Here are some general database tuning suggestions:

- Good database design — Distribute the database workload across multiple disks to avoid or reduce disk overloading. Good design also includes proper sizing and organization of tables, indexes, logs, and so on.
- Disk I/O optimization — Disk I/O optimization is related directly to throughput and scalability. Access to even the fastest disk is orders of magnitude slower than memory access. Whenever possible, optimize the number of disk accesses. In general, selecting a larger block/buffer size for I/O reduces the number of disk accesses and might substantially increase throughput in a heavily loaded production environment.
- Checkpointing — This mechanism periodically flushes all dirty cache data to disk, which increases the I/O activity and system resource usage for the duration of the checkpoint. Although frequent checkpointing can increase the consistency of on-disk data, it can also slow database performance. Most database systems have checkpointing capability, but not all database systems provide user-level controls. Oracle, for example, allows administrators to set the frequency of checkpoints while users have no control over SQLServer 7.x checkpoints. For recommended settings, see the product documentation for the database you are using.

Database-Specific Tuning

Here are some basic tuning suggestions for Oracle, SQL Server, and Sybase. Again, you should also check the tuning guidelines in your database-specific vendor documentation.

Oracle

This section describes performance tuning for Oracle 8.1.7.

- **Number of processes** — On most operating systems, each connection to the Oracle server spawns a shadow process to service the connection. Thus, the maximum number of processes allowed for the Oracle server must account for the number of simultaneous users, as well as the number of background processes used by the Oracle server. The default number is usually not big enough for a system that needs to support a large number of concurrent operations. For platform-specific issues, see your Oracle administrator's guide. The current setting of this parameter can be obtained with the following query:

```
SELECT name, value FROM v$parameter WHERE name = 'processes';
```

- **Shared pool size** — The shared pool is an important part of the Oracle server system global area (SGA). The SGA is a group of shared memory structures that contain data and control information for one Oracle database instance. If multiple users are concurrently connected to the same instance, the data in the instance's SGA is shared among the users. The shared pool portion of the SGA caches data for two major areas: the library cache and the dictionary cache. The library cache stores SQL-related information and control structures (for example, parsed SQL statement, locks). The dictionary cache stores operational metadata for SQL processing.

For most applications, the shared pool size is critical to Oracle performance. If the shared pool is too small, the server must dedicate resources to managing the limited amount of available space. This consumes CPU resources and causes contention because Oracle imposes restrictions on the parallel management of the various caches. The more you use triggers and stored procedures, the larger the shared pool must be. The `SHARED_POOL_SIZE` initialization parameter specifies the size of the shared pool in bytes.

The following query monitors the amount of free memory in the shared pool:

```
SELECT * FROM v$sgastat
WHERE name = 'free memory' AND pool = 'shared pool';
```

- **Maximum opened cursor** — To prevent any single connection taking all the resources in the Oracle server, the `OPEN_CURSORS` initialization parameter allows administrators to limit the maximum number of opened cursors for each connection. Unfortunately, the default value for this parameter is too small for systems such as WebLogic Server. Cursor information can be monitored using the following query:

```
SELECT name, value FROM v$sysstat
WHERE name LIKE 'opened cursor%';
```

- **Database block size** — A block is Oracle's basic unit for storing data and the smallest unit of I/O. One data block corresponds to a specific number of bytes of physical database

space on disk. This concept of a block is specific to Oracle RDBMS and should not be confused with the block size of the underlying operating system. Note that since the block size affects physical storage, this value can be set only during the creation of the database; it cannot be changed once the database has been created. The current setting of this parameter can be obtained with the following query:

```
SELECT name, value FROM v$parameter WHERE name = 'db_block_size';
```

- **Sort area size** — Increasing the sort area increases the performance of large sorts because it allows the sort to be performed in memory during query processing. This can be important, as there is only one sort area for each connection at any point in time. The default value of this `init.ora` parameter is usually the size of 6–8 data blocks. This value is usually sufficient for OLTP operations but should be increased for decision support operation, large bulk operations, or large index-related operations (for example, recreating an index). When performing these types of operations, you should tune the following `init.ora` parameters (which are currently set for 8K data blocks):

```
sort_area_size = 65536  
sort_area_retained_size = 65536
```

Microsoft SQL Server

The following guidelines pertain to performance tuning parameters for Microsoft SQL Server databases. For more information about these parameters, see your Microsoft SQL Server documentation.

- Store `tempdb` on a fast I/O device.
- Increase the recovery interval if `perfmon` shows an increase in I/O.
- Use an I/O block size larger than 2 KB.

Sybase

The following guidelines pertain to performance tuning parameters for Sybase databases. For more information about these parameters, see your Sybase documentation.

- Lower recovery interval setting results in more frequent checkpoint operations, resulting in more I/O operations.
- Use an I/O block size larger than 2 KB.
- Sybase controls the number of engines in a symmetric multiprocessor (SMP) environment. They recommend configuring this setting to equal the number of CPUs minus 1.

Identify the Best JVM Settings

Tune your JVM's heap garbage collection and heap size parameters to get the best performance out of your JVM. The Sun HotSpot and WebLogic JRockit JVM parameters that most significantly affect performance are listed below. For more detailed information, consult your JVM vendor's tuning documentation, as well as the JVM-related reading material at [“Java Virtual Machine \(JVM\) Information” on page A-8](#).

Sun JDK

When using the HotSpot VM option (`-server` or `-client`), experiment with the following garbage collection parameters:

- `-Xms` and `-Xmx` (use equal settings at start up)
- `-XX:NewSize` and `-XX:MaxNewSize`
- `-XX:SurvivorRatio`
- `-XX:+UseISM -XX:+AggressiveHeap`

For more information about tuning the HotSpot JVM, see [“JVM Heap Size and Garbage Collection” on page 3-3](#).

JRockit JDK

When using JRockit's JVM, experiment with the following garbage collection parameters:

- `-Xms` and `-Xmx` (use equal settings at startup)
- `-Xns`
- `-Xgc: parallel`
- `-XXenablefatspin`

Also, see [WebLogic JRockit Documentation](#).

Tune WebLogic Server Performance Parameters

The WebLogic Server configuration file (`config.xml`) contains a number of OOTB (out-of-the-box) performance-related parameters that can be fine-tuned depending on your environment and applications. Tuning these parameters based on your system requirements

(rather than running with default settings) can greatly improve both single-node performance and the scalability characteristics of an application.

Try experimenting with the following WebLogic Server configuration tuning parameters to determine your system's "sweet spot" for optimal performance:

- Modify the value of the execute queue's Thread Count, as described in ["Tuning the Default Execute Queue Threads" on page 4-6](#).
- If possible, use native performance packs (`NativeIOEnabled=true`), as described in ["Using WebLogic Server "Native IO" Performance Packs" on page 4-5](#).
- Use application-specific execute queues, as described in ["Using Execute Queues to Control Thread Usage" on page 6-5](#).
- When using a JDBC Connection Pool, modify the following attributes:
 - `DriverName`: Use the thin driver or `jdbcDriver`, as described in ["Using JDBC Drivers with WebLogic Server" in *Programming WebLogic JDBC*](#).
 - `InitialCapacity`: Set this to equal the `MaxCapacity` value, as described in ["Tuning JDBC Connection Pool Initial Capacity" on page 4-15](#).
 - `MaxCapacity`: Set the `MaxCapacity` value to at least equal the Thread Count value, and then, if necessary, increase it again until you find the right number, as described in ["Tuning JDBC Connection Pool Maximum Capacity" on page 4-15](#).
- Set the connection pool size to equal the execute queue's Thread Count, as described in ["How JDBC Connection Pools Enhance Performance" on page 4-14](#).
- Set the statement cache as described in ["Caching Prepared and Callable Statements" on page 4-16](#).
- Use multiple execute queues for servlets and JSPs, as described in ["Assigning Servlets and JSPs to Execute Queues" on page 6-8](#), and for EJBs and RMI, as described in ["Assigning EJBs and RMI Objects to Execute Queues" on page 6-9](#).
- Consider switching the default Java compiler for JSP compilation, `javac`, which is significantly slower than `jikes` or `sj`, as described in ["Setting Your Java Compiler" on page 4-16](#).

Monitor Disk and CPU Utilization

After following the previous steps, run your application under a high load while monitoring the:

- Application server (disk and CPU utilization)
- Database server (disk and CPU utilization)

To check your disk utilization on Solaris or Linux, use the `iostat -D <interval>` command, where the *interval* value determines how many seconds you want to elapse between monitoring cycles. To check your CPU utilization, simply leave off the `-D` flag (`iostat <interval>`).

For Windows, use the Performance Monitor tool (`perfmon`), to monitor both your disk and CPU utilization.

The goal is to get to a point where the application server becomes 100 percent utilized. If you find that the application server CPU is not close to 100 percent, confirm whether the database is bottlenecked. If the database CPU is 100 percent utilized, then check your application SQL calls query plans. For example, are your SQL calls using indexes or doing linear searches? Also, confirm whether there are too many `ORDER BY` clauses used in your application that are affecting the database CPU.

If you discover that the database disk is the bottleneck (for example, if the disk is 100 percent utilized), try moving to faster disks or to a RAID (redundant array of independent disks) configuration, assuming the application is not doing more writes than required.

Once you know the database server is *not* the bottleneck, determine whether the application server disk is the bottleneck. Some of the disk bottlenecks for application server disks are:

- JMS file store writes
- Transaction logging (tlogs)
- HTTP logging
- Server logging

The disk I/O on an application server can be optimized using faster disks or RAID, disabling synchronous JMS writes, using JTA direct writes for tlogs, or increasing the HTTP log buffer.

Monitor Data Transfers Across the Network

Check the amount of data transferred between the application and the application server, and between the application server and the database server. This amount should not exceed your network bandwidth; otherwise, your network becomes the bottleneck. To verify this, monitor the network statistics for retransmission and duplicate packets. This can be done using the following command:

```
netstat -s -P tcp
```

For instructions on viewing other TCP parameters using the `netstat -s -P` command, see [“Setting TCP Parameters With the `ndd` Command” on page 2-4](#).

Check For Frequent Standard I/O or Logging

Make sure your application is *not* doing too much standard I/O or excessive logging. Either situation could significantly slow system performance. In production environments, remove all `system.out.println` statements from your code, as these statements should only be used in development environments for debugging purposes.

Locate Bottlenecks in Your Applications

If you determine that neither the network nor the database server is the bottleneck, start looking at your WebLogic Server applications. Most importantly, is the machine running WebLogic Server able to get 100 percent CPU utilization with a high client load? If the answer is *no*, then check if there is any locking taking place in the application. You should profile your application using a commercially available tool (for example, JProbe or OptimizeIt) to pinpoint bottlenecks and improve application performance.

Tip: Even if you find that the CPU is 100 percent utilized, you should profile your application for performance improvements.

For more information about application profiling tools, see [“Using Performance Analysis Tools” on page 6-2](#).

Tune Your Application

This section contains recommended application-specific tuning suggestions for performance improvement.

EJBs

- Stateless session beans and MDBs (message-driven beans) — For maximum concurrency, the pool sizes should be at least as large as the thread count of the execute queue that handles requests to such beans.
- Use concurrency strategy.
- Experiment with EJB pool settings.

- Use Call-by-reference.
- Cache EJBs.
- Increase the MDB pool size for asynchronous message consumption.

See [Chapter 5, “Tuning WebLogic Server EJBs.”](#)

JSPs and Servlets

- Disable checks for JSP page checks and servlet reloading.
- Use in-memory session persistence, as described in [“Managing Session Persistence” on page 6-4.](#)
- Precompile JSPs, as described in [“Precompiling JSPs”](#) in *Programming WebLogic JSP*

See [“Introduction to Programming”](#) in *Programming WebLogic HTTP Servlets*.

JMS

- Avoid JMS message selectors and use multiple queues/topics to do message selection.
- Use asynchronous (`onMessage`) JMS consumers instead of synchronous receivers.
- Defer JMS acknowledgments and commits.

See the [“WebLogic JMS Performance Guide”](#) white paper on BEA dev2dev. For administrative tuning guidelines, see [“JMS Tuning”](#) in the *Administration Console Online Help*.

JDBC

- Tune your JDBC connection pool’s Initial Capacity and Max Capacity settings to complete database requests as fast as possible, rather than creating new connections.
- Cache prepared and callable statements used in your applications to minimize processing costs.
- Make your transactions single-batch by collecting a set of data operations and submitting an update transaction in one statement in the form.

See [“How JDBC Connection Pools Enhance Performance” on page 4-14](#) and [“Performance Tuning Your JDBC Application”](#) in *Programming WebLogic JDBC*.

Top Tuning Recommendations for WebLogic Server

Tuning Hardware, Operating System, and Network Performance

The following sections describe issues related to optimizing hardware, operating system, and network performance:

- “Hardware Tuning” on page 2-1
- “Operating System Tuning” on page 2-3
- “Network Performance” on page 2-8

Hardware Tuning

When you examine performance, a number of factors influence how much capacity a given hardware configuration will need in order to support WebLogic Server and a given application. The hardware capacity required to support your application depends on the specifics of the application and configuration. You should consider how each factor applies to your configuration and application. Before continuing with this section, here are some recommended starting points for planning your hardware configuration:

- **BEA WebLogic Server Capacity Planning Guide**, at <http://e-docs.bea.com/wls/docs81/capplan/index.html>, is a guide for *capacity planning* efforts for enterprise-level solutions built with WebLogic Server, with a focus on server hardware requirements.
- The **Standard Performance Evaluation Corporation**, at www.spec.org, provides a set of standardized benchmarks and metrics for evaluating computer system performance.

Supported Platforms

The following table provides some links to the information on the [Supported Configurations](#) pages, at <http://e-docs.bea.com/platform/suppconfigs/index.html>, which contains the latest certification information on the hardware/operating system platforms that are supported for each release of WebLogic Server.

Table 2-1 Platform-Specific Tuning Information

Platform	For more information
Bull/IBM pSeries with AIX	See the Bull/IBM links on the Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/index.html . <ul style="list-style-type: none">• Bull/IBM pSeries with AIX 5L v5.1• Bull/IBM pSeries with AIX 5L v5.2
Hewlett-Packard 9000 with HP-UX	See Hewlett-Packard HP/9000 with HP-UX 11.0 and 11i on the Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/hpux/index.html . See also “ Hewlett-Packard Company Information ” on page A-4 .
Intel Pentium-compatible with Windows	See the Intel/Windows links on the Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/index.html . <ul style="list-style-type: none">• Windows 2000 Server or Windows 2000 Advanced Server• Windows 2000 Professional• Windows XP See also “ Microsoft Information ” on page A-4 .

Table 2-1 Platform-Specific Tuning Information

Platform	For more information
Intel 32-bit-compatible with Red Hat Advanced Server	<p>See the Red Hat links on the Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/index.html.</p> <ul style="list-style-type: none"> • Red Hat Enterprise Linux AS 2.1 and ES 2.1 for IA-32 • Red Hat Enterprise Linux WS 2.1 for IA-32 <p>See also “Linux OS Information” on page A-3.</p>
Intel 64-bit-compatible with SuSE Linux	<p>See SuSE Linux (SLES 8) for IA-32 on the Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/index.html.</p> <p>See also “Linux OS Information” on page A-3.</p>
Sun Microsystems SPARC with Solaris	<p>See the Sun Microsystems SPARC Solaris links on the Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/index.html.</p> <ul style="list-style-type: none"> • SPARC with Solaris 8 • SPARC with Solaris 9 <p>See also “Sun Microsystems Information” on page A-2.</p>

Operating System Tuning

Tune your operating system according to your operating system documentation. BEA certifies WebLogic Server on multiple operating systems on the [Supported Configurations](#) pages, at <http://e-docs.bea.com/platform/suppconfigs/index.html>.

For Windows platforms, the default settings are usually sufficient. However, the Solaris and Linux platforms usually need to be tuned appropriately.

Solaris Tuning Parameters

The following sections provide information on tuning Solaris operating systems.

Setting TCP Parameters With the `ndd` Command

Set the following TCP-related tuning parameters using the `ndd` command, as demonstrated in the following example:

```
ndd -set /dev/tcp tcp_conn_req_max_q 16384
```

Table 2-2 Suggested TCP-Related Parameter Values

Parameter	Suggested Value
/dev/tcp tcp_time_wait_interval	60000
/dev/tcp tcp_conn_req_max_q	16384
/dev/tcp tcp_conn_req_max_q0	16384
/dev/tcp tcp_ip_abort_interval	60000
/dev/tcp tcp_keepalive_interval	7200000
/dev/tcp tcp_rexmit_interval_initial	4000
/dev/tcp tcp_rexmit_interval_max	10000
/dev/tcp tcp_rexmit_interval_min	3000
/dev/tcp tcp_smallest_anon_port	32768
/dev/tcp tcp_xmit_hiwat	131072
/dev/tcp tcp_recv_hiwat	131072
/dev/ce instance	0
/dev/ce rx_intr_time	32

Note: Prior to Solaris 2.7, the `tcp_time_wait_interval` parameter was called `tcp_close_wait_interval`. This parameter determines the time interval that a TCP socket is kept alive after issuing a close call. The default value of this parameter on Solaris is four minutes. When many clients connect for a short period of time, holding these socket resources can have a significant negative impact on performance. Setting this parameter to a value of 60000 (60 seconds) has shown a significant throughput enhancement when running benchmark JSP tests on Solaris. You might want to reduce this setting further if the server gets backed up with a queue of half-opened connections.

Tip: Use the `netstat -s -P tcp` command to view all available TCP parameters.

Setting Parameters In the `/etc/system` File

Each socket connection to the server consumes a file descriptor. To optimize socket performance, you need to configure your operating system to have the appropriate number of file descriptors. Therefore, you should change the default file descriptor limits, as well as the hash table size and other tuning parameters in the `/etc/system` file, to the recommended values in the following table.

Note: You must reboot your machine anytime you modify `/etc/system` parameters.

Table 2-3 Suggested `/etc/system` Values

Parameter	Suggested Value
<code>set rlim_fd_cur</code>	8192
<code>set rlim_fd_max</code>	8192
<code>set tcp:tcp_conn_hash_size</code>	32768
<code>set shmsys:shminfo_shmmax</code>	4294967295
Note: This should only be set for machines that have at least 4 GB RAM or higher.	
<code>set autoup</code>	900
<code>set tune_t_fsflushr</code>	1

CE Gigabit Network Card Settings

If you are using CE gigabit cards, we recommend using the following settings.

Table 2-4 Suggested CE Gigabit Card Values

Parameter	Suggested Value
<code>set ce:ce_bcopy_thresh</code>	256
<code>set ce:ce_dvma_thresh</code>	256

Table 2-4 Suggested CE Gigabit Card Values

Parameter	Suggested Value
set ce:ce_taskq_disable	1
set ce:ce_ring_size	256
set ce:ce_comp_ring_size	1024
set ce:ce_tx_ring_size	4096

For more information about Solaris tuning options, see:

- *Solaris Tunable Parameters Reference Manual* (Solaris 8), at <http://docs.sun.com/db/doc/816-0607>
- *Solaris Tunable Parameters Reference Manual* (Solaris 9), at <http://docs.sun.com/db/doc/806-7009>

Linux Tuning Parameters

For Linux operating systems, the following settings are recommended for optimal performance.

Table 2-5 Suggested Linux Values

Parameter	Suggested Value
/sbin/ifconfig lo mtu	1500
kernel.msgmni	1024
kernel.sem	1000 32000 32 512
fs.file-max	65535
kernel.shmmax	2147483648
net.ipv4.tcp_max_syn_backlog	8192

For more information about Linux tuning, you should consult your Linux vendor's documentation. Also, the [Ipsysctl Tutorial 1.0.4](#), at

<http://ipsysctl-tutorial.frozentux.net/ipsysctl-tutorial.html>, describes all of the IP options provided by Linux.

HP-UX Tuning Parameters

For HP-UX operating systems, the following TCP settings are recommended for optimal performance.

Table 2-6 Suggested HP-UX TCP Values

Parameter	Suggested Value
tcp_conn_req_max	4096
tcp_xmit_hiwater_def	1048576
tcp_ip_abort_interval	60000
tcp_rexmit_interval_initial	4000
tcp_keepalive_interval	900000

For more HP-UX tuning information, see the *Tunable Kernel Parameters* reference documentation, at <http://docs.hp.com/hpux/onlinedocs/TKP-90203/TKP-90203.html>.

Other Operating System Tuning Information

For more information about Windows, HP-UX, and AIX tuning options, refer to the following Web sites:

- For Windows tuning information, see the *Microsoft Windows 2000 TCP/IP Implementation Details white paper*, at http://www.microsoft.com/windows2000/techinfo/howitworks/communication/s/networkbasics/tcpip_implement.asp.
- For AIX tuning information, see the *AIX 5L Version 5.2 Performance Management Guide*, at http://publib16.boulder.ibm.com/pseries/en_US/aixbman/prftungd/prftungd.htm.

- Maximum memory for a user process — Check your operating system documentation for the maximum memory available for a user process. In some operating systems, this value is as low as 128 MB. Also, refer to your operating system documentation. For more information about memory management, see [Chapter 3, “Tuning Java Virtual Machines \(JVMs\).”](#)

Network Performance

Network performance is affected when the supply of resources is unable to keep up with the demand for resources. Today’s enterprise-level networks are very fast and are now rarely the direct cause of performance in well-designed applications. However, if you find that you have a problem with one or more network components (hardware or software), work with your network administrator to isolate and eliminate the problem. You should also verify that you have an appropriate amount of network bandwidth available for WebLogic Server and the connections it makes to other tiers in your architecture, such as client and database connections. Therefore, it is important to continually monitor your network performance to troubleshoot potential performance bottlenecks.

Determining Network Bandwidth

A common definition of bandwidth is “the rate of the data communications transmission, usually measured in bits-per-second, which is the capacity of the link to send and receive communications.” A machine running WebLogic Server requires enough network bandwidth to handle all WebLogic Server client connections. In the case of programmatic clients, each client JVM has a single socket to the server, and each socket requires dedicated bandwidth. A WebLogic Server instance handling programmatic clients should have 125–150 percent of the bandwidth that a similar Web server would handle. If you are handling only HTTP clients, expect a bandwidth requirement similar to a Web server serving static pages.

To determine whether you have enough bandwidth in a given deployment, you can use the network monitoring tools provided by your network operating system vendor to see what the load is on the network system. You can also use common operating system tools, such as the `netstat` command for Solaris or the System Monitor (`perfmon`) for Windows, to monitor your network utilization. If the load is very high, bandwidth may be a bottleneck for your system.

Also monitor the amount of data being transferred across the your network by checking the data transferred between the application and the application server, and between the application server and the database server. This amount should not exceed your network bandwidth; otherwise, your network becomes the bottleneck. To verify this, monitor the network statistics for retransmission and duplicate packets, as follows:

```
netstat -s -P tcp
```

For instructions on viewing other TCP parameters using the `netstat -s -P` command, see [“Setting TCP Parameters With the `ndd` Command” on page 2-4](#).

LAN Infrastructure

Your local area network must be fast enough to handle your application’s peak capacity. If your network is fully utilized, in that the amount of traffic consistently exceeds its bandwidth capacity, yet your WebLogic Server machine is not fully utilized, do one of the following:

- Redesign the network and redistribute the load.
- Reduce the number of network clients.
- Increase the number of systems handling the network load.

Tuning Java Virtual Machines (JVMs)

The Java virtual machine (JVM) is a virtual “execution engine” instance that executes the bytecodes in Java class files on a microprocessor. How you tune your JVM affects the performance of WebLogic Server and your applications.

The following sections discuss JVM tuning options for WebLogic Server:

- [“JVM Tuning Considerations” on page 3-2](#)
- [“Which JVM for Your System?” on page 3-3](#)
- [“JVM Heap Size and Garbage Collection” on page 3-3](#)
- [“Specifying Heap Size Values” on page 3-6](#)
- [“Automatically Logging Low Memory Conditions” on page 3-9](#)
- [“Manually Requesting Garbage Collection” on page 3-10](#)
- [“Setting Java HotSpot VM Options” on page 3-10](#)

JVM Tuning Considerations

Table 3-1 presents general JVM tuning considerations for WebLogic Server.

Table 3-1 General JVM Tuning Considerations

Tuning Factor	Information Reference
JVM vendor and version	Use only production JVMs on which WebLogic Server has been certified. WebLogic Server 8.1 supports only those JVMs that are J2SE 1.4-compliant. The Supported Configurations pages at http://e-docs.bea.com/platform/suppconfigs/index.html are frequently updated and contains the latest certification information on various platforms.
Tuning heap size and garbage collection	For WebLogic Server heap size tuning details, see “ JVM Heap Size and Garbage Collection ” on page 3-3.
Choosing a GC (garbage collection) scheme	Depending on your application, there are a number of GC schemes available for managing your system memory, as described in “ Choosing a Garbage Collection Scheme ” on page 3-4.
Mixed client/server JVMs	Deployments using different JVM versions for the client and server are supported in WebLogic Server. See the support page for Mixed Client/Server JVMs , at http://e-docs.bea.com/platform/suppconfigs/index.html#mix .
UNIX threading models	Choices you make about Solaris threading models can have a large impact on the performance of your JVM on Solaris. You can choose from multiple threading models and different methods of synchronization within the model, but this varies from JVM to JVM. See “ Performance Documentation For the Java Hotspot Virtual Machine: Threading ” on Sun Microsystems’ Web site at http://http://java.sun.com/docs/hotspot/theads/threads.html .

Which JVM for Your System?

Although this section focuses on Sun Microsystems' J2SE 1.4 JVM for the Windows, UNIX, and Linux platforms, the BEA WebLogic JRockit JVM was developed expressly for server-side applications and optimized for Intel architectures to ensure reliability, scalability, manageability, and flexibility for Java applications. For more information about the benefits of using JRockit on Windows and Linux platforms, see [WebLogic JRockit Documentation](http://e-docs.bea.com/more_jrockit.html), at http://e-docs.bea.com/more_jrockit.html.

For more information on JVMs in general, see the [Introduction](http://java.sun.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html#3057) to the JVM specification, at <http://java.sun.com/docs/books/vmspec/2nd-edition/html/Introduction.doc.html#3057>. For links to related reading for JVM tuning, see [Appendix A, "Related Reading: Performance Tools and Information."](#)

Changing To a Different JVM

When you create a domain, if you choose to customize the configuration, the Configuration Wizard presents a list of SDKs that WebLogic Server installed. From this list, you choose the JVM that you want to run your domain and the wizard configures the BEA start scripts based on your choice. After you create a domain, if you want to use a different JVM, you can modify the scripts as follows. For more information, see ["Changing the JVM That Runs Servers"](#) at <http://e-docs.bea.com/wls/docs81/adminguide/startstop.html#ChangingJVM>.

JVM Heap Size and Garbage Collection

Garbage collection is the JVM's process of freeing up unused Java objects in the Java heap. The Java heap is where the objects of a Java program live. It is a repository for live objects, dead objects, and free memory. When an object can no longer be reached from any pointer in the running program, it is considered "garbage" and ready for collection.

The JVM heap size determines how often and how long the VM spends collecting garbage. An acceptable rate for garbage collection is application-specific and should be adjusted after analyzing the actual time and frequency of garbage collections. If you set a large heap size, full garbage collection is slower, but it occurs less frequently. If you set your heap size in accordance with your memory needs, full garbage collection is faster, but occurs more frequently.

The goal of tuning your heap size is to minimize the time that your JVM spends doing garbage collection while maximizing the number of clients that WebLogic Server can handle at a given time. To ensure maximum performance during benchmarking, you might set high heap size values to ensure that garbage collection does not occur during the entire run of the benchmark.

You might see the following Java error if you are running out of heap space:

```
java.lang.OutOfMemoryError <<no stack trace available>>
java.lang.OutOfMemoryError <<no stack trace available>>
Exception in thread "main"
```

To modify heap space values, see [“Specifying Heap Size Values” on page 3-6](#).

To configure WebLogic Server to detect automatically when you are running out of heap space and to address low memory conditions in the server, see [“Automatically Logging Low Memory Conditions” on page 3-9](#).

Choosing a Garbage Collection Scheme

Depending on which JVM you are using, you can choose from several garbage collection schemes to manage your system memory. For example, some garbage collection schemes are more appropriate for a given type of application. Once you have an understanding of the workload of the application and the different garbage collection algorithms utilized by the JVM, you can optimize the configuration of the garbage collection.

Refer to the following links for in-depth discussions of garbage collection options for your JVM:

- For an overview of the garbage collection schemes available with Sun’s HotSpot VM, see [Tuning Garbage Collection with the 1.4.2 Java Virtual Machine](#), at <http://java.sun.com/docs/hotspot/gc1.4.2.index.htm>.
- For a comprehensive explanation of the collection schemes available with JDK 1.4.1, see [Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1](#), at <http://developers.sun.com/techtopics/mobility/midp/articles/garbagecollection2/index.html>
- For a discussion of the garbage collection schemes available with the BEA WebLogic JRockit JVM, see [WebLogic JRockit Documentation](#), at http://e-docs.bea.com/more_jrockit.html.
- For some pointers about garbage collection from an HP perspective, see [Tuning Garbage Collection with the 1.4.2 Java Virtual Machine](#), at http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,1604,00.html.

Using Verbose Garbage Collection to Determine Heap Size

The HotSpot VM's verbose garbage collection option (`verbosegc`) enables you to measure exactly how much time and resources are put into garbage collection. To determine the most effective heap size, turn on verbose garbage collection and redirect the output to a log file for diagnostic purposes.

The following steps outline this procedure:

1. Monitor the performance of WebLogic Server under maximum load while running your application.
2. Use the `-verbosegc` option to turn on verbose garbage collection output for your JVM and redirect *both* the standard error and standard output to a log file.

This places thread dump information in the proper context with WebLogic Server informational and error messages, and provides a more useful log for diagnostic purposes.

For example, on Windows and Solaris, enter the following:

```
% java -ms32m -mx200m -verbosegc -classpath $CLASSPATH
-Dweblogic.Name=%SERVER_NAME% -Dbea.home="C:\bea"
-Dweblogic.management.username=%WLS_USER%
-Dweblogic.management.password=%WLS_PW%
-Dweblogic.management.server=%ADMIN_URL%
-Dweblogic.ProductionModeEnabled=%STARTMODE%
-Djava.security.policy="%WL_HOME%\server\lib\weblogic.policy"
weblogic.Server
>> logfile.txt 2>&1
```

where the `logfile.txt 2>&1` command redirects both the standard error and standard output to a log file.

On HP/UX, use the following option to redirect `stderr` `stdout` to a single file:

```
-Xverbosegc:file=/tmp/gc$$$.out
```

where `$$` maps to the process ID (PID) of the Java process. Because the output includes timestamps for when garbage collection ran, you can infer how often garbage collection occurs.

3. Analyze the following data points:
 - a. How often is garbage collection taking place? In the `weblogic.log` file, compare the time stamps around the garbage collection.

- b. How long is garbage collection taking? Full garbage collection should not take longer than 3 to 5 seconds.
 - c. What is your average memory footprint? In other words, what does the heap settle back down to after each full garbage collection? If the heap always settles to 85 percent free, you might set the heap size smaller.
4. If you are using 1.4 Java HotSpot JVM, set the New generation heap sizes.

See “[Specifying Heap Size Values](#)” on page 3-6 and Table 3-2, “[Java Heap Size Options](#),” at [page 3-8](#).

Note: For information about setting the appropriate heap sizes for the BEA WebLogic JRockit JVM, see [WebLogic JRockit Documentation](#), at http://e-docs.bea.com/more_jrockit.html.

5. Make sure that the heap size is not larger than the available free RAM on your system.

Use as large a heap size as possible without causing your system to “swap” pages to disk. The amount of free RAM on your system depends on your hardware configuration and the memory requirements of running processes on your machine. See your system administrator for help in determining the amount of free RAM on your system.

6. If you find that your system is spending too much time collecting garbage (your allocated “virtual” memory is more than your RAM can handle), lower your heap size.

Typically, you should use 80 percent of the available RAM (not taken by the operating system or other processes) for your JVM.

7. If you find that you have a large amount of available free RAM remaining, run more instances of WebLogic Server on your machine.

Remember, the goal of tuning your heap size is to minimize the time that your JVM spends doing garbage collection while maximizing the number of clients that WebLogic Server can handle at a given time.

Note: For information about using the BEA WebLogic JRockit JVM `-Xgcreport` option to print a comprehensive garbage collection report at program completion, see [WebLogic JRockit Documentation](#), at http://e-docs.bea.com/more_jrockit.html.

Specifying Heap Size Values

You must specify Java heap size values each time you start an instance of WebLogic Server. This can be done either from the `java` command line or by modifying the default values in the sample

startup scripts that are provided with the WebLogic distribution for starting WebLogic Server, as explained in [“Using WebLogic Startup Scripts to Set Heap Size” on page 3-7](#).

For example, when you start a WebLogic Server instance from a `java` command line, you could specify the HotSpot VM heap size values as follows:

```
$ java -XX:NewSize=128m -XX:MaxNewSize=128m -XX:SurvivorRatio=8 -Xms512m
-Xmx512m
```

The default size for these values is measured in bytes. Append the letter ‘k’ or ‘K’ to the value to indicate kilobytes, ‘m’ or ‘M’ to indicate megabytes, and ‘g’ or ‘G’ to indicate gigabytes. The example above allocates 128 megabytes of memory to the New generation and maximum New generation heap sizes, and 512 megabytes of memory to the minimum and maximum heap sizes for the WebLogic Server instance running in the JVM. For more information on the heap size options, see [“Java Heap Size Options” on page 3-8](#).

Using WebLogic Startup Scripts to Set Heap Size

Sample startup scripts are provided with the WebLogic Server distribution for starting the server and for setting the environment to build and run the server:

- `startWLS.cmd` and `setEnv.cmd` (Windows).
- `startWLS.sh` and `setEnv.sh` (UNIX and Windows. On Windows, these scripts support the MKS and Cygnus BASH UNIX shell emulators.)

If you used the Configuration Wizard to create your domain, the WebLogic startup scripts are located in the *domain-name* directory where you specified your domain. By default, this directory is `BEA_HOME\user_projects\domain\domain-name`, where `BEA_HOME` is the directory that contains the product installation, and *domain-name* is the name of the domain directory defined by the selected configuration template. For more information about creating domains using the Configuration Wizard, see [“Creating Domains Using the Configuration Wizard”](#) at <http://edocs.bea.com/platform/docs81/configwiz/creatdom.html>.

The WebLogic startup scripts set environment variables, such as the default memory arguments passed to Java (that is, heap size) and the location of the JDK, and then start the JVM with WebLogic Server arguments. Be aware that the WebLogic Server startup scripts specify default heap size parameters; therefore, you need to modify them to fit your environment and applications. For instructions on how to modifying the startup scripts to set the Java heap size options, see [“Specifying Java Options for a WebLogic Server Instance”](#) at <http://e-docs.bea.com/wls/docs81/adminguide/startstop.html#JavaOptions>.

Java Heap Size Options

You achieve best performance by individually tuning each application. However, configuring the Java HotSpot VM heap size options listed in [Table 3-2](#) when starting WebLogic Server increases performance for most applications.

These options may differ depending on your architecture and operating system. See your vendor’s documentation for platform-specific JVM tuning options.

Table 3-2 Java Heap Size Options

Task	Option	Recommended Value
Setting the New generation heap size	-XX:NewSize	<p>Set this value to a multiple of 1024 that is greater than 1MB. As a general rule, set -XX:NewSize to be one-fourth the size of the maximum heap size. Increase the value of this option for larger numbers of short-lived objects.</p> <p>Be sure to increase the New generation as you increase the number of processors. Memory allocation can be parallel, but garbage collection is not parallel.</p>
Setting the maximum New generation heap size	-XX:MaxNewSize	<p>Set this value to a multiple of 1024 that is greater than 1MB.</p>
Setting New heap size ratios	-XX:SurvivorRatio	<p>The New generation area is divided into three sub-areas: Eden, and two survivor spaces that are equal in size.</p> <p>Configure the ratio of the Eden/survivor space size. Try setting this value to 8, and then monitor your garbage collection.</p>
Setting minimum heap size	-Xms	<p>Set the minimum size of the memory allocation pool. Set this value to a multiple of 1024 that is greater than 1MB. As a general rule, set minimum heap size (-Xms) equal to the maximum heap size (-Xmx) to minimize garbage collections.</p>
Setting maximum heap size	-Xmx	<p>Set the maximum size of the memory allocation pool. Set this value to a multiple of 1024 that is greater than 1MB.</p>

Note: For information about setting the appropriate heap sizes for WebLogic's JRockit JVM, see [WebLogic JRockit Documentation](http://e-docs.bea.com/more_jrockit.html), at http://e-docs.bea.com/more_jrockit.html.

Automatically Logging Low Memory Conditions

WebLogic Server enables you to automatically log low memory conditions observed by the server. WebLogic Server detects low memory by sampling the available free memory a set number of times during a time interval. At the end of each interval, an average of the free memory is recorded and compared to the average obtained at the next interval. If the average drops by a user-configured amount after any sample interval, the server logs a low memory warning message in the log file and sets the server health state to “warning.”

If the average free memory ever drops below 5 percent of the initial free memory recorded immediately after you start the server, WebLogic Server logs a message to the log file.

You configure each aspect of the low memory detection process using the Administration Console:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the navigation tree to display the servers configured in your domain.
4. Click the name of the server instance that you want to configure. Note that you configure low memory detection on a per-server basis.
5. Select the Configuration → Tuning tab in the right pane.
6. On the Advanced Options bar, click Show to display additional attributes.
7. Modify the following Memory Option attributes as necessary to tune low memory detection for the selected server instance:
 - Low Memory GCThreshold: Enter a percentage value (0–99 percent) to represent the threshold after which WebLogic Server logs a low memory warning and changes the health state to “warning.” By default, Memory GCThreshold is set to 5 percent. This means that by default the server logs a low memory warning after the average free memory reaches 5 percent of the initial free memory measured at the server's boot time.

- **Low Memory Sample Size:** Enter the number of times the server samples free memory during a fixed time period. By default, the server samples free memory 10 times each interval to acquire the average free memory. Using a higher sample size can increase the accuracy of the reading.
 - **Low Memory Time Interval:** Enter the time, in seconds, that define the interval over which the server determines average free memory values. By default WebLogic Server obtains an average free memory value every 3600 seconds.
8. Click **Apply** to apply your changes.
 9. Reboot the server to use the new low memory detection attributes.

Manually Requesting Garbage Collection

Make sure that full garbage collection is necessary before manually selecting it on a server. When you perform garbage collection, the JVM often examines every living object in the heap.

To use the Administration Console to request garbage collection on a specific server instance:

1. On the Administration Console, expand the server instance node for the server whose memory usage you want to view. A dialog box in the right pane shows the tabs associated with this instance.
2. Select the **Monitoring** → **Performance** tab.
3. Check the **Memory Usage** graph for high usage. Note that the **Memory Usage** graph displays information only for a server that is currently running.
4. Click the **Force garbage collection** button to request garbage collection. The **Force garbage collection** button calls the JVM's `System.gc()` method to perform garbage collection. Note, however, that the JVM implementation itself decides whether or not the request actually triggers garbage collection.

Setting Java HotSpot VM Options

You can use standard `java` command-line options to improve the performance of your JVM. How you use these options depends on how your application is coded. Although command-line options are consistent across platforms, some platforms may have different defaults.

Test both your client and server JVMs to see which options perform better for your particular application. The Sun Microsystems [Java HotSpot VM Options](#) document provides information on the command-line options and environment variables that can affect the performance

characteristics of the Java HotSpot Virtual Machine. See <http://java.sun.com/docs/hotspot/VMOptions.html>.

For a brief discussion of additional “non-standard” VM options that can also affect performance, see “[Non-Standard HotSpot VM Options for Windows, Solaris, and Linux](#)” on page 3-12.

Note: For information about using the WebLogic JRockit JVM command-line options to improve performance, see [WebLogic JRockit Documentation](#), at http://e-docs.bea.com/more_jrockit.html.

Standard HotSpot VM Options for Windows, Solaris, and Linux

You can use standard `java` options to improve performance on Windows, Solaris, and Linux platforms. These options are supported on the current Java the run-time environment and will also be supported in future releases of HotSpot. When specifying one of the standard options listed [Table 3-3](#) through the `java` command, WebLogic Server invokes a particular version of the JVM.

Table 3-3 Standard Java HotSpot VM Options

Option	Platform	Description
<code>-client</code>	Windows Solaris	Selects the Java HotSpot Client VM. This is the default for Windows and Solaris.
<code>-server</code>	Windows Solaris Linux	Selects the Java HotSpot Server VM.
<code>-hotspot</code>	Linux	Selects the Java HotSpot Client VM. This is the default for Linux.

For additional examples of the standard HotSpot VM options, see:

- [Standard Options for Windows \(Win32\) VMs](#) at <http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/java.html#standard>.
- [Standard Options for Solaris VMs](#) at <http://java.sun.com/j2se/1.4.1/docs/tooldocs/solaris/java.html#standard>.
- [Standard Options for Linux VMs](#) at <http://java.sun.com/j2se/1.4.1/docs/tooldocs/linux/java.html#standard>.

Sun Microsystems' [Java Virtual Machine](http://java.sun.com/j2se/1.4.1/docs/guide/vm/index.html) document provides a detailed discussion of the Client and Server implementations of the Java virtual machine for J2SE 1.4. See <http://java.sun.com/j2se/1.4.1/docs/guide/vm/index.html>.

Non-Standard HotSpot VM Options for Windows, Solaris, and Linux

You can also use non-standard `java` options to improve performance. How you use these options depends on how your application is coded. Although command-line options are consistent across platforms, some platforms may have different defaults. Note that non-standard command-line options are subject to change in future releases.

For examples of the non-standard options for improving performance on the Hotspot VM on Windows, Solaris, and Linux, see:

- [Non-Standard Options for Windows VMs \(Win32\)](http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/java.html#nonstandard) at <http://java.sun.com/j2se/1.4.1/docs/tooldocs/windows/java.html#nonstandard>.
- [Non-Standard Options for Solaris VMs](http://java.sun.com/j2se/1.4.1/docs/tooldocs/solaris/java.html#nonstandard) at <http://java.sun.com/j2se/1.4.1/docs/tooldocs/solaris/java.html#nonstandard>.
- [Non-Standard Options for Linux VMs](http://java.sun.com/j2se/1.4.1/docs/tooldocs/linux/java.html#nonstandard) at <http://java.sun.com/j2se/1.4.1/docs/tooldocs/linux/java.html#nonstandard>.

Tuning WebLogic Server

The following sections describe how to tune WebLogic Server to match your application needs.

- [“Setting Java Parameters for Starting WebLogic Server” on page 4-1](#)
- [“Setting Performance-Related Configuration Parameters” on page 4-2](#)
- [“Development vs. Production Mode Default Tuning Values” on page 4-4](#)
- [“Using WebLogic Server “Native IO” Performance Packs” on page 4-5](#)
- [“Tuning the Default Execute Queue Threads” on page 4-6](#)
- [“Tuning Connection Backlog Buffering” on page 4-14](#)
- [“How JDBC Connection Pools Enhance Performance” on page 4-14](#)
- [“Setting Your Java Compiler” on page 4-16](#)
- [“Using WebLogic Server Clusters to Improve Performance” on page 4-18](#)
- [“Monitoring a WebLogic Server Domain” on page 4-21](#)

Setting Java Parameters for Starting WebLogic Server

Java parameters must be specified whenever you start WebLogic Server. For simple invocations, this can be done from the command line with the `weblogic.Server` command. However, because the arguments needed to start WebLogic Server from the command line can be lengthy and prone to error, BEA recommends that you incorporate the command into a script. To simply this process, you can modify the default values in the sample scripts that are provided with the

WebLogic distribution to start WebLogic Server, as described in “[Specifying Java Options for a WebLogic Server Instance](#)” at

<http://e-docs.bea.com/wls/docs81/adminguide/startstop.html#JavaOptions>.

If you used the Configuration Wizard to create your domain, the WebLogic startup scripts are located in the *domain-name* directory where you specified your domain. By default, this directory is *BEA_HOME*\user_projects\domain*domain-name*, where *BEA_HOME* is the directory that contains the product installation, and *domain-name* is the name of the domain directory defined by the selected configuration template. For more information about creating domains using the Configuration Wizard, see “[Creating Domains Using the Configuration Wizard](#)” at <http://e-docs.bea.com/wls/docs81/http://edocs.bea.com/platform/docs81/configwiz/creatdom.html>.

You need to modify some default Java values in these scripts to fit your environment and applications. The important performance tuning parameters in these files are the *JAVA_HOME* parameter and the Java heap size parameters:

- Change the value of the variable *JAVA_HOME* to the location of your JDK. For example:

```
set JAVA_HOME=C:\bea\jdk141_03
```

- For higher performance throughput, set the minimum java heap size equal to the maximum heap size. For example:

```
"%JAVA_HOME%\bin\java" -hotspot -Xms512m -Xmx512m -classpath  
%CLASSPATH% -
```

See “[Specifying Heap Size Values](#)” on page 3-6 for details about setting heap size options.

Setting Performance-Related Configuration Parameters

The WebLogic Server configuration file (*config.xml*) contains a number of performance-related parameters that can be fine-tuned depending on your environment and applications. Tuning these parameters based on your system requirements (rather than running with default settings) can greatly improve both single-node performance and the scalability characteristics of an application.

Within a WebLogic Server domain, the configuration file is located on the machine that hosts the Administration Server, and provides persistent storage of WebLogic MBean attribute values. The Administration Server serves as a central point of contact for server instances and system administration tools. A domain may also include additional WebLogic Server instances called Managed Servers, which are used mainly for servicing applications.

When the Administration Server starts, it reads the domain configuration file and overrides the default attribute values of the administration MBeans with any attribute values found in the configuration file. Every time you change an attribute using the system administration tools (using either the command-line interface or the Administration Console), its value is stored in the appropriate administration MBean and written to the configuration file.

For more information about system administration infrastructure, see “[Overview of WebLogic Server System Administration](#)” in the *Administration Guide* at

<http://e-docs.bea.com/wls/docs81/adminguide/overview.html>.

Overview of WebLogic Server System Administration

[Table 4-1](#) lists the `config.xml` file parameters that affect server performance.

Table 4-1 Performance-Related `config.xml` Elements

Element	Attributes	Console Field	For information
Server	NativeIOEnabled	Native IO Enabled	See “ Using WebLogic Server “Native IO” Performance Packs ” on page 4-5.
ExecuteQueue	ThreadCount	Thread Count	See “ Tuning the Default Execute Queue Threads ” on page 4-6.
ExecuteQueue	QueueLength	Queue Length	See “ Tuning Execute Queues for Overflow Conditions ” on page 4-10.
	QueueLengthThresholdPercent	Queue Length Threshold Percent	
	ThreadsIncrease	Threads Increase	
	ThreadsMaximum	Threads Maximum	
	Thread Priority	Thread Priority	
Server	StuckThreadMaxTime	Stuck Thread Max Time	See “ Tuning the Execute Thread Detection Behavior ” on page 4-12.
	StuckThreadTimerInterval	Stuck Thread Timer Interval	
Server	ThreadPoolPercent SocketReaders	Socket Readers	See “ Allocating Execute Threads to Act as Socket Readers ” on page 4-9.

Table 4-1 Performance-Related config.xml Elements (Continued)

Element	Attributes	Console Field	For information
Server	AcceptBacklog	Accept Backlog	See “ Tuning Connection Backlog Buffering ” on page 4-14.
JDBCConnectionPool	InitialCapacity MaxCapacity	Initial Capacity Max Capacity	See “ How JDBC Connection Pools Enhance Performance ” on page 4-14.
JDBCConnectionPool	StatementCacheSize	Statement Cache Size	See “ Caching Prepared and Callable Statements ” on page 4-16.

Development vs. Production Mode Default Tuning Values

You can indicate whether a domain is to be used in a development environment or a production environment. WebLogic Server uses different default values for various services depending on the type of environment you specify.

[Table 4-2](#) lists the performance-related configuration parameters that differ when switching from development to production startup mode.

Table 4-2 Development and Production Startup Mode Tuning Defaults

Tuning Parameter	Development Mode Default	Production Mode Default
Execute Queue: ThreadCount	15 threads	25 threads
JDBC Connection Pool: MaxCapacity	15 connections	25 connections

The tuning defaults discussed in throughout *WebLogic Performance and Tuning Guide* refer to the “development mode” defaults, which is the default startup mode when WebLogic Server is installed. For information on switching the startup mode from development to production, see [Changing the Runtime Mode](#) in the *Administration Console Online Help* at <http://e-docs.bea.com/wls/docs81/ConsoleHelp.server.html#ChangingRuntimeMode>.

For a complete listing of the differences between development and production startup modes, see the “[Differences Between Configuration Startup Modes](#)” section in *Creating WebLogic Configurations Using the Configuration Wizard*, at <http://edocs.bea.com/platform/docs81/configwiz/newdom.htm#devprod>.

Using WebLogic Server “Native IO” Performance Packs

Benchmarks show major performance improvements when you use native performance packs on machines that host WebLogic Server instances. Performance packs use a platform-optimized, native socket multiplexor to improve server performance. For example, the native socket reader multiplexor threads have their own execute queue and do not borrow threads from the default execute queue, which frees up default execute threads to do application work.

However, if you must use the pure-Java socket reader implementation for host machines, you can still improve the performance of socket communication by configuring the proper number of socket reader threads for each server instance and client machine.

Which Platforms Have Performance Packs?

To see which supported platforms currently have performance packs available:

1. Go to [Supported Configurations for WebLogic Server](http://e-docs.bea.com/platform/suppconfigs/index.html) at <http://e-docs.bea.com/platform/suppconfigs/index.html>.
2. From the list of supported configurations, click on the link for the platform that you need.
The ensuing page contains tables of information for each supported WebLogic Server releases (including service packs). Within each release table there is a “Performance Pack” entry that indicates whether a performance pack is “Included” in the release.
3. To verify performance pack information, you can either click on a specific WebLogic Server release at the top of the page and scan the corresponding table, or use your browser’s Edit → Find feature to search for all instances of “Performance Pack” on the page.

Enabling Performance Packs

The use of native performance packs are enabled by default in the `config.xml` shipped with your distribution. To verify this setting in your configuration file, check that the `NativeIOEnabled` attribute of the `Server` element is set to “true” (`NativeIOEnabled=true`).

You can also use the Administration Console to verify that performance packs are enabled:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Click the name of the server instance that you want to configure.
5. Select the Configuration → Tuning tab.
6. If the Enable Native IO check box is not selected, select the check box.
7. Click Apply.
8. Restart the server.

Tuning the Default Execute Queue Threads

The value of the `ThreadCount` attribute of an `ExecuteQueue` element in the `config.xml` file equals the number of simultaneous operations that can be performed by applications that use the execute queue. As work enters an instance of WebLogic Server, it is placed in an execute queue. This work is then assigned to a thread that does the work on it. Threads consume resources, so handle this attribute with care—you can degrade performance by increasing the value unnecessarily.

By default, a new WebLogic Server instance is configured with a development mode execute queue, `weblogic.kernel.default`, that contains 15 threads. In addition, WebLogic Server provides two other pre-configured queues:

- `weblogic.admin.HTTP`—Available only on Administration Servers, this queue is reserved for communicating with the Administration Console; you cannot reconfigure it.
- `weblogic.admin.RMI`—Both Administration Servers and Managed Servers have this queue; it is reserved for administrative traffic; you cannot reconfigure it.

Unless you configure additional execute queues, and assign applications to them, Web applications and RMI objects use `weblogic.kernel.default`.

Note: If native performance packs are not being used for your platform, you may need to tune the default number of execute queue threads *and* the percentage of threads that act as socket readers to achieve optimal performance. For more information, see [“Allocating Execute Threads to Act as Socket Readers” on page 4-9](#).

Should You Modify the Default Thread Count?

Adding more threads to the default execute queue does not necessarily imply that you can process more work. Even if you add more threads, you are still limited by the power of your processor. Because threads consume memory, you can degrade performance by increasing the value of the `ThreadCount` attribute unnecessarily. A high execute thread count causes more memory to be used and increases context switching, which can degrade performance.

The value of the `ThreadCount` attribute depends very much on the type of work your application does. For example, if your client application is thin and does a lot of its work through remote invocation, that client application will spend more time connected — and thus will require a higher thread count — than a client application that does a lot of client-side processing.

If you do not need to use more than 15 threads (the development default) or 25 threads (the production default) for your work, do not change the value of this attribute. As a general rule, if your application makes database calls that take a long time to return, you will need more execute threads than an application that makes calls that are short and turn over very rapidly. For the latter case, using a smaller number of execute threads could improve performance.

Scenarios for Modifying the Default Thread Count

To determine the ideal thread count for an execute queue, monitor the queue's throughput while all applications in the queue are operating at maximum load. Increase the number of threads in the queue and repeat the load test until you reach the optimal throughput for the queue. (At some point, increasing the number of threads will lead to enough context switching that the throughput for the queue begins to decrease.)

Note: The WebLogic Server Administration Console displays the cumulative throughput for all of a server's execute queues. To access this throughput value, follow steps 1-6 in [“Modifying the Default Thread Count”](#) on page 4-9.

[Table 4-3](#) shows default scenarios for adjusting available threads in relation to the number of CPUs available in the WebLogic Server domain. These scenarios also assume that WebLogic Server is running under maximum load, and that all thread requests are satisfied by using the default execute queue. If you configure additional execute queues and assign applications to specific queues, monitor results on a pool-by-pool basis.

Table 4-3 Scenarios for Modifying the Default Thread Count

When...	Results	Do This:
Thread Count < number of CPUs	Your thread count is too low if: <ul style="list-style-type: none">• CPU is waiting to do work, but there is work that could be done.• Cannot get 100 percent CPU utilization rate.	Increase the thread count.
Thread Count = number of CPUs	Theoretically ideal, but the CPUs are still under-utilized.	Increase the thread count.
Thread Count > number of CPUs (by a moderate number of threads)	Practically ideal, with a moderate amount of context switching and a high CPU utilization rate.	Tune the moderate number of threads and compare performance results.
Thread Count > number of CPUs (by a large number of threads)	Too much context switching, which can lead to significant performance degradation. Your performance may increase as you decrease the number of threads.	<p>Reduce the number of threads so that it equals the number of CPUs, and then add <i>only</i> the number of “stuck” threads that you have determined.</p> <p>For example, if you have four processors, then four threads can be running concurrently with the number of stuck threads. So, you want the execute threads to be 4 + the number of stuck threads.</p> <p>To determine the amount of stuck threads, see “Tuning the Execute Thread Detection Behavior” on page 4-12.</p> <p>Note: This recommendation is highly application-dependent. For instance, the length of time the application might block threads can invalidate the formula.</p>

Modifying the Default Thread Count

To modify the default execute queue thread count using the Administration Console:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Right-click the name of the server instance that contains the execute queue you want to configure, and then select View Execute Queues on the pop-up menu to display a table of execute queues that can be modified.

Note: You can only modify the default execute queue for the server or a user-defined execute queue.

5. In the Name column, click directly on the default execute queue name to display the Configuration tab for modifying execute queues.
6. Locate the Thread Count value and increase or decrease it, as appropriate.
7. Click Apply to save your changes.
8. Reboot the selected server to enable the new execute queue settings.

Assigning Applications to Execute Queues

Although you can configure the default execute queue to supply the optimal number threads for all WebLogic Server applications, configuring multiple execute queues can provide additional control for key applications. By using multiple execute queues, you can guarantee that selected applications have access to a fixed number of execute threads, regardless of the load on WebLogic Server. See [“Using Execute Queues to Control Thread Usage” on page 6-5](#) for more information on assigning applications to configured execute queues.

Allocating Execute Threads to Act as Socket Readers

For best socket performance, BEA recommends that you use the native socket reader implementation, rather than the pure-Java implementation, on machines that host WebLogic Server instances (see [“Using WebLogic Server “Native IO” Performance Packs” on page 4-5](#)). However, if you must use the pure-Java socket reader implementation for host machines, you can still improve the performance of socket communication by configuring the proper number of execute threads to act as socket reader threads for each server instance and client machine.

The `ThreadPoolPercentSocketReaders` attribute sets the maximum percentage of execute threads that are set to read messages from a socket. The optimal value for this attribute is application-specific. The default value is 33, and the valid range is 1–99.

Allocating execute threads to act as socket reader threads increases the speed and the ability of the server to accept client requests. It is essential to balance the number of execute threads that are devoted to reading messages from a socket and those threads that perform the actual execution of tasks in the server.

Setting the Number of Socket Reader Threads For a Server Instance

To use the Administration Console to set the maximum percentage of execute threads that read messages from a socket:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Click the name of the server that you want to configure.
5. Select the Configuration → Tuning tab.
6. Edit the percentage of Java reader threads in the Socket Readers attribute field. The number of Java socket readers is computed as a percentage of the number of total execute threads (as shown in the Thread Count field for the Execute Queue).
7. Apply the changes.

Setting the Number of Socket Reader Threads on Client Machines

On client machines, you can configure the number of available socket reader threads in the JVM that runs the client. Specify the socket readers by defining the following parameters in the `java` command line for the client:

```
-Dweblogic.ThreadPoolSize=value  
-Dweblogic.ThreadPoolPercentSocketReaders=value
```

Tuning Execute Queues for Overflow Conditions

You can configure WebLogic Server to detect and optionally address potential overflow conditions in the default execute queue or any user-defined execute queue. WebLogic Server considers a queue to have a possible overflow condition when its current size reaches a

user-defined percentage of its maximum size. When this threshold is reached, the server changes its health state to “warning” and can optionally allocate additional threads to perform the outstanding work in the queue, thereby reducing its size.

To automatically detect and address overflow conditions in a queue, you configure the following items:

- The threshold at which the server indicates an overflow condition. This value is set as a percentage of the configured size of the execute queue (the `QueueLength` value).
- The number of threads to add to the execute queue when an overflow condition is detected. These additional threads work to reduce the size of the queue to its normal operating size.
- The maximum number of threads available to the queue. In particular, setting the maximum number of threads prevents the server from assigning an overly high thread count in response to overload conditions.

To tune an execute queue using the WebLogic Server Administration Console:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Right-click the name of the server instance that contains the execute queue you want to configure, and then select View Execute Queues from the pop-up menu to display a table of execute queues that can be modified.

Note: You can only modify the default execute queue for the server or a user-defined execute queue.

5. In the Name column, directly click the default execute queue name (or the user-defined execute queue) that you want to configure.
6. On the execute queue Configuration tab, specify how the server instance should detect an overflow condition for the selected queue by modifying the following attributes:
 - `Queue Length`: Specifies the maximum number of simultaneous requests that the server can hold in the queue. The default of 65536 requests represents a very large number of requests; outstanding requests in the queue should rarely, if ever reach this maximum value. Always leave the Queue Length at the default value of 65536 entries.
 - `Queue Length Threshold Percent`: The percentage (from 1–99) of the Queue Length size that can be reached before the server indicates an overflow condition for the queue. All actual queue length sizes below the threshold percentage are considered

- normal; sizes above the threshold percentage indicate an overflow. By default, the Queue Length Threshold Percent is set to 90 percent.
- `Thread Priority`: The priority of the threads associated with the queue. By default, the Thread Priority is set to 5.
7. To specify how this server should address an overflow condition for the selected queue, modify the following attribute:
 - `Threads Increase`: The number of threads WebLogic Server should add to this execute queue when it detects an overflow condition. If you specify zero threads (the default), the server changes its health state to “warning” in response to an overflow condition in the execute queue, but it does not allocate additional threads to reduce the workload.
 8. To limit the maximum number of threads that can be added to the selected queue, modify the following attribute:
 - `Threads Maximum`: The maximum number of threads that this execute queue can have; this value prevents WebLogic Server from creating an overly high thread count in the queue in response to continual overflow conditions. By default, Threads Maximum is set to 400.
 9. Click Apply to save your changes.
 10. Reboot the selected server to enable the new execute queue settings.

Tuning the Execute Thread Detection Behavior

WebLogic Server automatically detects when a thread in an execute queue becomes “stuck.” Because a stuck thread cannot complete its current work or accept new work, the server logs a message each time it diagnoses a stuck thread. If all threads in an execute queue become stuck, the server changes its health state to either “warning” or “critical” depending on the execute queue:

- If all threads in the default queue become stuck, the server changes its health state to “critical.” (You can set up the Node Manager application to automatically shut down and restart servers in the critical health state. For more information, see “[Node Manager Capabilities](http://e-docs.bea.com/wls/docs81/adminguide/nodemgr.html#NodeManagerCapabilities)” in *Configuring and Managing WebLogic Server* at <http://e-docs.bea.com/wls/docs81/adminguide/nodemgr.html#NodeManagerCapabilities>.)
- If all threads in `weblogic.admin.HTTP`, `weblogic.admin.RMI`, or a user-defined execute queue become stuck, the server changes its health state to “warning.”

WebLogic Server diagnoses a thread as stuck if it is continually working (not idle) for a set period of time. You can tune a server's thread detection behavior by changing the length of time before a thread is diagnosed as stuck, and by changing the frequency with which the server checks for stuck threads.

Note: Although you can change the criteria WebLogic Server uses to determine whether a thread is stuck, you cannot change the default behavior of setting the “warning” and “critical” health states when all threads in a particular execute queue become stuck. For more information, see [Overview of WebLogic Logging Services in Using WebLogic Logging Services](#).

To configure WebLogic Server stuck thread detection behavior:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Click the name of the server instance that you want to modify for improved stuck thread detection.

Note: You configure stuck thread detection parameters on a per-server basis, rather than on a per-execute queue basis.

5. Select the Configuration → Tuning tab in the right pane.
6. Modify the following attributes as necessary to tune thread detection behavior for the server:
 - **Stuck Thread Max Time:** Enter the number of seconds, that a thread must be continually working before this server diagnoses the thread as being stuck. By default, WebLogic Server considers a thread to be “stuck” after 600 seconds of continuous use.
 - **Stuck Thread Timer Interval:** Enter the number of seconds, after which WebLogic Server periodically scans threads to see if they have been continually working for the length of time specified by **Stuck Thread Max Time**. By default, WebLogic Server sets this interval to 600 seconds.
7. Click Apply to save your changes.
8. Reboot the server to use the new settings.

Tuning Connection Backlog Buffering

Use the `AcceptBacklog` attribute of the `Server` element in the `config.xml` file to set the number of connection requests the WebLogic Server instance will accept before refusing additional requests. The `AcceptBacklog` attribute specifies how many Transmission Control Protocol (TCP) connections can be buffered in a wait queue. This fixed-size queue is populated with requests for connections that the TCP stack has received, but the application has not accepted yet. The default value is 50 and the maximum value is operating system dependent.

To tune the Accept Backlog value from the Administration Console:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Click the name of the server instance that you want to configure.
5. Select the Configuration → Tuning tab.
6. Modify the default Accept Backlog value as necessary to tune how many TCP connections can be buffered in a wait queue:
 - During operations, if many connections are dropped or refused at the client, and no other error messages are on the server, the Accept Backlog value might be set too low.
 - If you are getting “connection refused” messages when you try to access WebLogic Server, raise the Accept Backlog value from the default by 25 percent. Continue increasing the value by 25 percent until the messages cease to appear.
7. Click Apply to save your changes.

How JDBC Connection Pools Enhance Performance

Establishing a JDBC connection with a DBMS can be very slow. If your application requires database connections that are repeatedly opened and closed, this can become a significant performance issue. WebLogic connection pools offer an efficient solution to the problem.

When WebLogic Server starts, connections from the connection pools are opened and are available to all clients. When a client closes a connection from a connection pool, the connection is returned to the pool and becomes available for other clients; the connection itself is not closed. There is little cost to opening and closing pool connections.

How many connections should you create in the pool? A connection pool can grow and shrink according to configured parameters, between a minimum and a maximum number of connections. The best performance occurs when the connection pool has as many connections as there are concurrent client sessions.

In addition to the following subsections, see “[Performance Tuning Your JDBC Application](#)” in *Programming WebLogic JDBC* at

<http://e-docs.bea.com/wls/docs81/jdbc/performance.html>.

Tuning JDBC Connection Pool Initial Capacity

The `InitialCapacity` attribute of the `JDBCConnectionPool` element enables you to set the number of physical database connections to create when configuring the pool. If the server cannot create this number of connections, the creation of this connection pool will fail.

During development, it may be convenient to set the value of the `InitialCapacity` attribute to a low number to help the server start up faster. In production systems, consider setting the `InitialCapacity` value equal to the `MaxCapacity` attribute’s default production mode setting of 25. This way, all database connections are acquired during server start-up. And if you need to tune the `MaxCapacity` value, make sure to set the `InitialCapacity` so that it equals the `MaxCapacity` value.

If `InitialCapacity` is less than `MaxCapacity`, the server needs to create additional database connections when its load is increased. When the server is under load, all resources should be working to complete requests as fast as possible, rather than creating new database connections.

Tuning JDBC Connection Pool Maximum Capacity

The `MaxCapacity` attribute of the `JDBCConnectionPool` element allows you to set the maximum number of physical database connections that a connection pool can contain. Different JDBC drivers and database servers might limit the number of possible physical connections.

The default settings for development and production mode are equal to the default number of execute threads: 15 for development mode; 25 for production mode. However, in production, it is advisable that the number of connections in the pool equal the number of concurrent client sessions that require JDBC connections. The pool capacity is independent of the number of execute threads in the server. There may be many more ongoing user sessions than there are execute threads.

Caching Prepared and Callable Statements

When you use a prepared statement or callable statement in an application or EJB, there is considerable processing overhead for the communication between the application server and the database server and on the database server itself. To minimize the processing costs, WebLogic Server can cache prepared and callable statements used in your applications. When an application or EJB calls any of the statements stored in the cache, WebLogic Server reuses the statement stored in the cache. Reusing prepared and callable statements reduces CPU usage on the database server, improving performance for the current statement and leaving CPU cycles for other tasks. For more details, see “[Increasing Performance with the Statement Cache](#)” in the *Administration Console Online Help* at

http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html#statementcache.

Using the statement cache can dramatically increase performance, but you must consider its limitations before you decide to use it. For more details, see “[Usage Restrictions for the Statement Cache](#)” in the *Administration Console Online Help* at

http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html#cache_restrictions.

Setting Your Java Compiler

The standard Java compiler for compiling JSP servlets is `javac`. You can improve performance significantly by setting your server’s java compiler to `sj` or `jikes` instead of `javac`. The following sections discuss this procedure and other compiler considerations.

Changing Compilers in the Administration Console

To change your compiler in the Administration Console:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Click the name of the server instance that you want to configure.
5. Select the Configuration → General tab and enter the full path of the compiler in the Java Compiler field. For example:

```
c:\visualcafe31\bin\sj.exe
```

6. Click Show on the Advanced Options bar to display additional attributes.
7. Enter the full path to the JRE `rt.jar` library in the Append to the Classpath field. For example:
`BEA_HOME\jdk141_02\jre\lib\rt.jar`
8. Click Apply.
9. Restart your server for the new Java Compiler and Append to Classpath values to take effect.

Setting Your Compiler in weblogic.xml

In the `weblogic.xml` file, the `jsp-descriptor` element defines parameter names and values for servlet JSPs.

- Use the `compileCommand` parameter to specify the Java compiler for compiling the generated JSP servlets.
- Use the `precompile` parameter to configure WebLogic Server to precompile your JSPs when WebLogic Server starts up.

For more information about setting your server's java compiler in the `weblogic.xml` file, see the [jsp-descriptor](http://e-docs.bea.com/wls/docs81/webapp/weblogic_xml.html#jsp-descriptor) element at http://e-docs.bea.com/wls/docs81/webapp/weblogic_xml.html#jsp-descriptor.

Compiling EJB Container Classes

Use the `weblogic.appc` utility to compile EJB 2.0 and 1.1 container classes. If you compile Jar files for deployment into the EJB container, you must use `weblogic.appc` to generate the container classes. By default, `ejbc` uses the `javac` compiler. For faster performance, specify a different compiler (such as Symantec `sj`) using the `-compiler` flag.

For more information, see “[Implementing EJBs](#)” in *Programming WebLogic Enterprise JavaBeans*.

Compiling on UNIX

If you receive the following error message received when compiling JSP files on a UNIX machine:

```
failed: java.io.IOException: Not enough space
```

Try any or all of the following solutions:

- Add more RAM if you have only 256 MB.
- Raise the file descriptor limit, for example:

```
set rlim_fd_max = 4096
set rlim_fd_cur = 1024
```
- Use the `-native` flag to use native threads when starting the JVM.

Using WebLogic Server Clusters to Improve Performance

A WebLogic Server cluster is a group of WebLogic Servers instances that together provide fail-over and replicated services to support scalable high-availability operations for clients within a domain. A cluster appears to its clients as a single server but is in fact a group of servers acting as one to provide increased scalability and reliability.

A domain can include multiple WebLogic Server clusters and non-clustered WebLogic Server instances. Clustered WebLogic Server instances within a domain behave similarly to non-clustered instances, except that they provide failover and load balancing. The Administration Server for the domain manages all the configuration parameters for the clustered and non-clustered instances.

For more information about clusters, see “[Introduction to WebLogic Server Clustering](http://e-docs.bea.com/wls/docs81/cluster/overview.html)” at <http://e-docs.bea.com/wls/docs81/cluster/overview.html>.

Scalability and High Availability

Scalability is the ability of a system to grow in one or more dimensions as more resources are added to the system. Typically, these dimensions include (among other things), the number of concurrent users that can be supported and the number of transactions that can be processed in a given unit of time.

Given a well-designed application, it is entirely possible to increase performance by simply adding more resources. To increase the load handling capabilities of WebLogic Server, add another WebLogic Server instance to your cluster—without changing your application. Clusters provide two key benefits that are not provided by a single server: scalability and availability.

WebLogic Server clusters bring scalability and high-availability to J2EE applications in a way that is transparent to application developers. Scalability expands the capacity of the middle tier beyond that of a single WebLogic Server or a single computer. The only limitation on cluster membership is that all WebLogic Servers must be able to communicate by IP multicast. New WebLogic Servers can be added to a cluster dynamically to increase capacity.

A WebLogic Server cluster guarantees high-availability by using the redundancy of multiple servers to insulate clients from failures. The same service can be provided on multiple servers in a cluster. If one server fails, another can take over. The ability to have a functioning server take over from a failed server increases the availability of the application to clients.

Caution: Provided that you have resolved all application and environment bottleneck issues, adding additional servers to a cluster should provide linear scalability. When doing benchmark or initial configuration test runs, isolate issues in a single server environment before moving to a clustered environment.

How to Ensure Scalability for WebLogic Clusters

In general, any operation that requires communication between the servers in a cluster is a potential scalability hindrance. The following sections provide information on issues that impact the ability to linearly scale clustered WebLogic servers:

- [“Database Bottlenecks” on page 4-19](#)
- [“Session Replication” on page 4-19](#)
- [“Invalidation of Entity EJBs” on page 4-20](#)
- [“Invalidation of HTTP sessions” on page 4-20](#)
- [“JNDI Binding, Unbinding and Rebinding” on page 4-20](#)

Database Bottlenecks

In many cases where a cluster of WebLogic servers fails to scale, the database is the bottleneck. In such situations, the only solutions are to tune the database or reduce load on the database by exploring other options. See [“JDBC Application Tuning” on page 6-2](#).

Session Replication

User session data can be stored in two standard ways in a J2EE application: stateful session EJBs or HTTP sessions. By themselves, they are rarely a impact cluster scalability. However, when coupled with a session replication mechanism required to provide high-availability, bottlenecks are introduced. If a J2EE application has Web and EJB components, you should store user session data in HTTP sessions rather than stateful session EJBs as HTTP session management provides more replication options than stateful session EJBs. See [“Managing Sessions” on page 6-4](#).

Invalidation of Entity EJBs

This applies to entity EJBs that use a concurrency strategy of `Optimistic` or `ReadOnly` with a read-write pattern.

Optimistic—When an `Optimistic` concurrency bean is updated, the EJB container sends a multicast message to other cluster members to invalidate their local copies of the bean. This is done to avoid optimistic concurrency exceptions being thrown by the other servers and hence the need to retry transactions. If updates to the EJBs are frequent, the work done by the servers to invalidate each other's local caches become a serious bottleneck.

ReadOnly with a read-write pattern—In this pattern, persistent data that would otherwise be represented by a single EJB are actually represented by two EJBs: one read-only and the other updateable. When the state of the updateable bean changes, the container automatically invalidates corresponding read-only EJB instance. If updates to the EJBs are frequent, the work done by the servers to invalidate the read-only EJBs becomes a serious bottleneck.

Invalidation of HTTP sessions

Similar to [“Invalidation of Entity EJBs” on page 4-20](#), HTTP sessions can also be invalidated. This is not as expensive as entity EJB invalidation, since only the session data stored in the secondary server needs to be invalidated. BEA advises users to not invalidate sessions unless absolutely required.

JNDI Binding, Unbinding and Rebinding

In general, JNDI binds, unbinds and rebinds are expensive operations. However, these operations become a bigger bottleneck in clustered environments because JNDI tree changes have to be propagated to all members of a cluster. If such operations are performed too frequently, they can reduce cluster scalability significantly.

Performance Considerations When Running Multiple Server Instances on Multi-CPU Machines

With multi-processor machines, additional consideration must be given to the ratio of the number of available CPUs to clustered WebLogic Server instances. Because WebLogic Server has no built-in limit to the number of server instances that reside in a cluster, large, multi-processor servers, such as Sun Microsystems' Sun Enterprise 10000, can potentially host very large clusters or multiple clusters.

In order to determine the optimal ratio of CPUs to WebLogic server instances, you must first ensure that an application is truly CPU-bound, rather than network or disk I/O-bound. Use the following steps to determine the optional ratio of CPUs to server instances:

1. Test your application to determine the *Network Requirements*.

If you discover that an application is primarily network I/O-bound, consider measures to increase network throughput before increasing the number of available CPUs. For truly network I/O-bound applications, installing a faster network interface card (NIC) may increase performance more than additional CPUs, because most CPUs would remain idle while waiting to read available sockets.

2. Test your application to determine the *Disk I/O Requirements*.

If you discover that an application is primarily disk I/O-bound, consider upgrading the number of disk spindles or individual disks and controllers before allocating additional CPUs.

3. Begin performance tests using a ratio of one WebLogic Server instance for every available CPU.
4. If CPU utilization is consistently at or near 100 percent, increase the ratio of CPUs to servers by adding an additional CPU. Add additional CPUs until utilization reaches an acceptable level. Remember, always reserve some spare CPU cycles on your production systems to perform any administration tasks that may occur.

Monitoring a WebLogic Server Domain

The tool for monitoring the health and performance of your WebLogic Server domain is the Administration Console. Using the Administration Console, you can view status and statistics for WebLogic Server resources such as servers, HTTP, the JTA subsystem, JNDI, security, CORBA connection pools, EJB, JDBC, and JMS.

For more details, see “[Monitoring a WebLogic Server Domain](#)” in *Configuring and Managing WebLogic Server* at

<http://e-docs.bea.com/wls/docs81/adminguide/monitoring.html>.

For example, there is a Server → Monitoring → Performance tab on the Administration Console that provides performance metrics related to pending and processed requests for the current server instance. It includes the following information:

- The number of idle threads assigned to the queue.
- The oldest pending request in the queue.

- Throughput, as measured by the number of requests already processed by the queue.
- Queue Length, as measured by the number waiting requests in the queue.
- The amount of memory available in the JVM heap.

For more details, see “[Server -> Monitoring -> Performance](#)” in the *Administration Console Online Help* at

http://e-docs.bea.com/wls/docs81/ConsoleHelp/domain_server_monitoring_performance.html.

Tuning WebLogic Server EJBs

The following sections describe how to tune WebLogic Server EJBs to match your application needs:

- “Setting Performance-Related `weblogic-ejb-jar.xml` Parameters” on page 5-1
- “Setting Performance-Related `weblogic-cmp-jar.xml` Parameters” on page 5-6
- “Tuning In Response to Monitoring Statistics” on page 5-7
- “Other Performance Improvement Strategies” on page 5-11

Setting Performance-Related `weblogic-ejb-jar.xml` Parameters

The `weblogic-ejb-jar.xml` deployment file contains the WebLogic Server-specific deployment elements that determine the concurrency, caching, and clustering behaviors of EJBs. It also contains deployment elements that map available WebLogic Server resources to EJBs. WebLogic Server resources include JDBC data sources and connection pools, JMS connection factories, and other deployed EJBs.

For information on how to modify the `weblogic-ejb-jar.xml` deployment file, see “[Editing Deployment Descriptors](#)” in *Programming WebLogic Enterprise JavaBeans* at <http://e-docs.bea.com/wls/docs81/ejb/implementing.html#EditingDeploymentDescriptors>.

Table 5-1 lists the `weblogic-ejb-jar.xml` file parameters that affect performance.

Table 5-1 Performance-Related `weblogic-ejb-jar.xml` Elements

Element	For more information
<code>max-beans-in-free-pool</code>	See “Setting EJB Pool Size for Session and Message-Driven Beans” on page 5-2.
<code>initial-beans-in-free-pool</code>	See “Tuning Pool Size for Stateless Sessions Beans at Startup” on page 5-4.
<code>max-beans-in-cache</code>	See “Setting Caching Size for Stateful Session and Entity Beans” on page 5-4.
<code>concurrency-strategy</code>	See “Deferring Database Locking” on page 5-5.
<code>isolation-level</code>	See “Setting Transaction Isolation Level” on page 5-6.

The following sections describe these elements.

Setting EJB Pool Size for Session and Message-Driven Beans

WebLogic Server maintains a free pool of EJBs for every stateless session bean class. The `max-beans-in-free-pool` element of the `weblogic-ejb-jar.xml` file defines the size of this pool.

In addition to this section, see [max-beans-in-free-pool](#) in *Programming WebLogic Enterprise JavaBeans* at <http://e-docs.bea.com/wls/docs81/ejb/DDreference-ejb-jar.html#max-beans-in-free-pool>.

When EJBs are created, the session bean instance is created and given an identity. When the client removes a bean, the bean instance is placed in the free pool. When you create a subsequent bean, you can avoid object allocation by reusing the previous instance that is in the free pool. The `max-beans-in-free-pool` element can improve performance if EJBs are frequently created and removed.

The EJB container creates new instances of message beans as needed for concurrent message processing. The `max-beans-in-pool` element puts an absolute limit on how many of these instances will be created. The container may override this setting according to the runtime resources that are available.

For the best performance for stateless session and message beans, use the default setting `max-beans-in-free-pool` element. The default allows you to run beans in parallel, using as many threads as possible.

Do not change the value of the `max-beans-in-free-pool` parameter for session beans unless you frequently create session beans, do a quick operation, and then throw them away. If you do this, enlarge your free pool by 25 to 50 percent and see if performance improves. If object creation represents a small fraction of your workload, increasing this parameter will not significantly improve performance. For applications where EJBs are database intensive, do not change the value of this parameter.

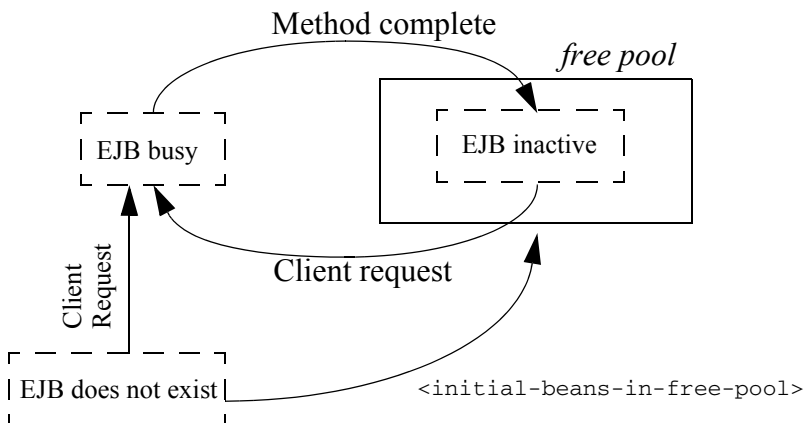
Caution: Tuning this parameter too high uses extra memory. Tuning it too low causes unnecessary object creation. If you are in doubt about changing this parameter, leave it unchanged.

Using a Free Pool to Improve Stateless Session Bean Performance

WebLogic Server uses a *free pool* to improve performance and throughput for stateless session EJBs. The free pool stores *unbound* stateless session EJBs. Unbound EJB instances are instances of a stateless session EJB class that are not processing a method call.

The following figure illustrates the WebLogic Server free pool, and the processes by which stateless EJBs enter and leave the pool. Dotted lines indicate the “state” of the EJB from the perspective of WebLogic Server.

Figure 5-1 WebLogic Server free pool showing stateless session EJB life cycle



Allocating Pool Size for Entity Beans

A pool of anonymous entity beans (that is., beans without a primary key class) used to invoke finders and home methods, and to create other entity beans. The `max-beans-in-free-pool` element also controls the size of this pool.

If you are running many finders or home methods or creating many beans, you may want to tune the `max-beans-in-free-pool` element so that there are enough beans available for use in the pool.

Tuning Pool Size for Stateless Sessions Beans at Startup

Use the `initial-beans-in-free-pool` element of the `weblogic-ejb-jar.xml` file to specify the number of stateless session bean instances in the free pool at startup.

If you specify a value for `initial-beans-in-free-pool`, WebLogic Server populates the free pool with the specified number of bean instances at startup. Populating the free pool in this way improves initial response time for the EJB, because initial requests for the bean can be satisfied without generating a new instance.

`initial-beans-in-free-pool` defaults to 0 if the element is not defined.

The `initial-beans-in-free-pool` element is described in *Programming WebLogic Enterprise JavaBeans* at

<http://e-docs.bea.com/wls/docs81/ejb/DDreference-ejb-jar.html#initial-beans-in-free-pool>.

Setting Caching Size for Stateful Session and Entity Beans

You can configure the number of active bean instances that are present in the EJB cache (the in-memory space where beans exist).

Use the `max-beans-in-cache` element of the `weblogic-ejb-jar.xml` file to specify the maximum number of objects of this bean class that are allowed in memory. When `max-beans-in-cache` is reached, WebLogic Server passivates some EJBs that have not been recently used by a client. The `max-beans-in-cache` element also affects when EJBs are removed from the WebLogic Server cache.

For more information, see “[Choosing a Concurrency Strategy](#)” in *Programming WebLogic Enterprise JavaBeans* at

<http://e-docs.bea.com/wls/docs81/ejb/entity.html#ChoosingaConcurrencyStrategy>.

The `max-beans-in-cache` element is described in *Programming WebLogic Enterprise JavaBeans* at <http://e-docs.bea.com/wls/docs81/ejb/DDreference-ejb-jar.html#max-beans-in-cache>.

Activation and Passivation of Stateful Session EJBs

Set the appropriate cache size with the `max-beans-in-cache` element to avoid excessive passivation and activation. Activation is the transfer of an EJB instance from secondary storage to memory. Passivation is the transfer of an EJB instance from memory to secondary storage. Tuning `max-beans-in-cache` too high consumes memory unnecessarily.

The EJB container performs passivation when the cache becomes full. When the EJB session object is needed again, the bean is activated by the container.

The container automatically manages this working set of session objects in the EJB cache without the client's or server's direct intervention. Specific callback methods in each EJB describe how to passivate (store in cache) or activate (retrieve from cache) these objects. Excessive activation and passivation nullifies the performance benefits of caching the working set of session objects in the EJB cache—especially when the application has to handle a large number of session objects.

Deferring Database Locking

WebLogic Server supports database locking, optimistic locking and exclusive locking mechanisms. The default and *recommended* mechanism for EJB 1.1 and EJB 2.0 is database locking.

Database locking improves concurrent access to entity EJBs. The WebLogic Server container improves concurrent access by deferring locking services to the underlying database. Unlike exclusive locking, with deferred database locking, the underlying data store can provide finer granularity for locking EJB data, in most cases, and provide deadlock detection.

For details about database locking, see [Choosing a Concurrency Strategy](#) in *Enterprise JavaBeans* at <http://e-docs.bea.com/wls/docs81/ejb/entity.html#ChoosingaConcurrencyStrategy>.

You specify the locking mechanism used for an EJB by setting the `concurrency-strategy` element in the `weblogic-ejb-jar.xml` file. See <http://e-docs.bea.com/wls/docs81/ejb/DDreference-ejb-jar.html#concurrency-strategy>.

Setting Transaction Isolation Level

Data accessibility is controlled through the transaction isolation level mechanism. Transaction isolation level determines the degree to which multiple interleaved transactions are prevented from interfering with each other in a multi-user database system. Transaction isolation is achieved through use of locking protocols that guide the reading and writing of transaction data. This transaction data is written to the disk in a process called “serialization.” Lower isolation levels give you better database concurrency at the cost of less transaction isolation.

For more information, see the description of the [isolation-level element](#) of the `weblogic-ejb-jar.xml` file in *Programming WebLogic Enterprise JavaBeans* at <http://e-docs.bea.com/wls/docs81/ejb/DDreference-ejb-jar.html#isolation-level>.

Refer to your database documentation for more information on the implications and support for different isolation levels.

Setting Performance-Related `weblogic-cmp-jar.xml` Parameters

[Table 5-1](#) lists the `weblogic-cmp-jar.xml` file parameters that affect performance.

Table 5-2 Performance-Related `weblogic-cmp-jar.xml` Parameters

Element	For more information
<code>relationship-caching</code>	See Relationship Caching at http://e-docs.bea.com/wls/docs81/ejb/entity.html#RelationshipCaching .
<code>order-database-operations</code>	See Database Operation Ordering at http://e-docs.bea.com/wls/docs81/ejb/entity.html#OrderingandBatchingOperations .
<code>enable-batch-operations</code>	See Database Operation Ordering at http://e-docs.bea.com/wls/docs81/ejb/entity.html#OrderingandBatchingOperations .

Table 5-2 Performance-Related weblogic-cmp-jar.xml Parameters (Continued)

Element	For more information
delay-database-insert-until	See Delaying Database Inserts at http://e-docs.bea.com/wls/docs81/ejb/entity.html#DelayingDatabaseInserts .
field-group	See Field Groups at http://e-docs.bea.com/wls/docs81/ejb/entity.HTML#FieldGroups .

Tuning In Response to Monitoring Statistics

The WebLogic Server Administration Console reports a wide variety of EJB runtime monitoring statistics, many of which are useful for tuning your EJBs. This section discusses how some of these statistics can help you tune the performance of EJBs.

To display the statistics in the Administration Console, see “[Monitoring EJBs](#)” at <http://e-docs.bea.com/wls/docs81/ConsoleHelp/ejb.html#MonitoringEJBs>.

Note: If you prefer to write a custom monitoring application, you can access the monitoring statistics programmatically by calling the relevant runtime MBeans. See the [Javadoc for the `weblogic.management.runtime` package](#).

Cache Miss Ratio

The cache miss ratio is a ratio of the number of times a container cannot find a bean in the cache (cache miss) to the number of times it attempts to find a bean in the cache (cache access):

$$\text{Cache Miss Ratio} = (\text{Cache Total Miss Count} / \text{Cache Total Access Count}) * 100$$

A high cache miss ratio could be indicative of an improperly sized cache. If your application uses a certain subset of beans (read primary keys) more frequently than others, it would be ideal to size your cache large enough so that the commonly used beans can remain in the cache as less commonly used beans are cycled in and out upon demand. If this is the nature of your application, you may be able to decrease your cache miss ratio significantly by increasing the maximum size of your cache.

If your application doesn’t necessarily use a subset of beans more frequently than others, increasing your maximum cache size may not affect your cache miss ratio. We recommend testing your application with different maximum cache sizes to determine which give the lowest

cache miss ratio. It is also important to keep in mind that your server has a finite amount of memory and therefore there is always a trade-off to increasing your cache size.

Lock Waiter Ratio

The lock waiter ratio of the number of times a thread had to wait to obtain a lock on a bean to the total amount of lock requests issued:

`Lock Waiter Ratio = (Current Waiter Count / Current Lock Entry Count) * 100`

A high lock waiter ratio can indicate a suboptimal concurrency strategy for the bean. If acceptable for your application, a concurrency strategy of Database or Optimistic will allow for more parallelism than an Exclusive strategy and remove the need for locking at the EJB container level.

Because locks are generally held for the duration of a transaction, reducing the duration of your transactions will free up beans more quickly and may help reduce your lock waiter ratio. To reduce transaction duration, avoid grouping large amounts of work into a single transaction unless absolutely necessary.

Lock Timeout Ratio

The lock timeout ratio of timeouts to accesses for the lock manager:

`Lock Timeout Ratio = (Lock Manager Timeout Total Count / Lock Manager Total Access Count) * 100`

The lock timeout ratio is closely related to the lock waiter ratio. If you are concerned about the lock timeout ratio for your bean, first take a look at the lock waiter ratio and our recommendations for reducing it (including possibly changing your concurrency strategy). If you can reduce or eliminate the number of times a thread has to wait for a lock on a bean, you will also reduce or eliminate the amount of timeouts that occur while waiting.

A high lock timeout ratio may also be indicative of an improper transaction timeout value. The maximum amount of time a thread will wait for a lock is equal to the current transaction timeout value.

If the transaction timeout value is set too low, threads may not be waiting long enough to obtain access to a bean and timing out prematurely. If this is the case, increasing the `trans-timeout-seconds` value for the bean may help reduce the lock timeout ratio.

Take care when increasing the `trans-timeout-seconds`, however, because doing so can cause threads to wait longer for a bean and threads are a valuable server resource. Also, doing so may increase the request time, as a request may wait longer before timing out.

Pool Miss Ratio

The pool miss ratio is a ratio of the number of times a request was made to get a bean from the pool when no beans were available, to the total number of requests for a bean made to the pool:

$\text{Pool Miss Ratio} = (\text{Pool Total Miss Count} / \text{Pool Total Access Count}) * 100$

If your pool miss ratio is high, you must determine what is happening to your bean instances. There are three things that can happen to your beans.

- They are in use.
- They were destroyed.
- They were removed.

Follow these steps to diagnose the problem:

1. Check your destroyed bean ratio to verify that bean instances are not being destroyed.

Investigate the cause and try to remedy the situation.

2. Examine the demand for the EJB, perhaps over a period of time.

One way to check this is via the Beans in Use Current Count and Idle Beans Count displayed in the Administration Console. If demand for your EJB spikes during a certain period of time, you may see a lot of pool misses as your pool is emptied and unable to fill additional requests.

As the demand for the EJB drops and beans are returned to the pool, many of the beans created to satisfy requests may be unable to fit in the pool and are therefore removed. If this is the case, you may be able to reduce the number of pool misses by increasing the maximum size of your free pool. This may allow beans that were created to satisfy demand during peak periods to remain in the pool so they can be used again when demand once again increases.

Destroyed Bean Ratio

The destroyed bean ratio is a ratio of the number of beans destroyed to the total number of requests for a bean.

$\text{Destroyed Bean Ratio} = (\text{Total Destroyed Count} / \text{Total Access Count}) * 100$

To reduce the number of destroyed beans, BEA recommends against throwing non-application exceptions from your bean code except in cases where you want the bean instance to be destroyed. A non-application exception is an exception that is either a `java.rmi.RemoteException`

(including exceptions that inherit from `RemoteException`) or is not defined in the throws clause of a method of an EJB's home or component interface.

In general, you should investigate which exceptions are causing your beans to be destroyed as they may be hurting performance and may indicate problem with the EJB or a resource used by the EJB.

Pool Timeout Ratio

The pool timeout ratio is a ratio of requests that have timed out waiting for a bean from the pool to the total number of requests made:

$$\text{Pool Timeout Ratio} = (\text{Pool Total Timeout Count} / \text{Pool Total Access Count}) * 100$$

A high pool timeout ratio could be indicative of an improperly sized free pool. Increasing the maximum size of your free pool via the `max-beans-in-free-pool` setting will increase the number of bean instances available to service requests and may reduce your pool timeout ratio.

Another factor affecting the number of pool timeouts is the configured transaction timeout for your bean. The maximum amount of time a thread will wait for a bean from the pool is equal to the default transaction timeout for the bean. Increasing the `trans-timeout-seconds` setting in your `weblogic-ejb-jar.xml` file will give threads more time to wait for a bean instance to become available.

Users should exercise caution when increasing this value, however, since doing so may cause threads to wait longer for a bean and threads are a valuable server resource. Also, request time might increase because a request will wait longer before timing out.

Transaction Rollback Ratio

The transaction rollback ratio is the ratio of transactions that have rolled back to the number of total transactions involving the EJB:

$$\text{Transaction Rollback Ratio} = (\text{Transaction Total Rollback Count} / \text{Transaction Total Count}) * 100$$

Begin investigating a high transaction rollback ratio by examining the [Transaction Timeout Ratio](#) reported in the Administration Console. If the transaction timeout ratio is higher than you expect, try to address the timeout problem first.

An unexpectedly high transaction rollback ratio could be caused by a number of things. We recommend investigating the cause of transaction rollbacks to find potential problems with your application or a resource used by your application.

Transaction Timeout Ratio

The transaction timeout ratio is the ratio of transactions that have timed out to the total number of transactions involving an EJB:

```
Transaction Timeout Ratio = (Transaction Total Timeout Count / Transaction
Total Count) * 100
```

A high transaction timeout ratio could be caused by the wrong transaction timeout value. For example, if your transaction timeout is set too low, you may be timing out transactions before the thread is able to complete the necessary work. Increasing your transaction timeout value may reduce the number of transaction timeouts.

You should exercise caution when increasing this value, however, since doing so can cause threads to wait longer for a resource before timing out. Also, request time might increase because a request will wait longer before timing out.

A high transaction timeout ratio could be caused by a number of things such as a bottleneck for a server resource. We recommend tracing through your transactions to investigate what is causing the timeouts so the problem can be addressed.

Other Performance Improvement Strategies

- [“Application-Level Caching” on page 5-11](#)
- [“Batch Operations” on page 5-12](#)
- [“Distributing Transactions Across EJBs in a WebLogic Server Cluster” on page 5-12](#)
- [“Using a Free Pool to Improve Stateless Session Bean Performance” on page 5-3](#)
- [“Indexing with a Version Column” on page 5-12](#)

Application-Level Caching

Application-level caching allows you to configure a single cache for use with multiple entity beans. This will help solve usability and performance problems. Previously, you were required to configure a separate cache for each entity bean that was part of an application. For more

information on combined caching, see [Application-Level Versus Bean-Level Caching](http://e-docs.bea.com/wls/docs81/ejb/entity.html#Application-LevelversusBean-LevelCaching) at <http://e-docs.bea.com/wls/docs81/ejb/entity.html#Application-LevelversusBean-LevelCaching>.

Batch Operations

Batch inserts, updates and deletes improve the performance of container-managed persistence (CMP) bean creation by enabling the EJB container to perform multiple database operations for CMP beans in one SQL statement, thereby reducing network roundtrips. For more information on batch operations, see “[Ordering and Batching Operations](http://e-docs.bea.com/wls/docs81/ejb/entity.html#OrderingandBatchingOperations)” at <http://e-docs.bea.com/wls/docs81/ejb/entity.html#OrderingandBatchingOperations>.

Distributing Transactions Across EJBs in a WebLogic Server Cluster

WebLogic Server provides additional transaction performance benefits for EJBs that reside in a WebLogic Server cluster. When a single transaction uses multiple EJBs, WebLogic Server attempts to use EJB instances from a single WebLogic Server instance, rather than using EJBs from different servers. This approach minimizes network traffic for the transaction.

In some cases, a transaction can use EJBs that reside on multiple WebLogic Server instances in a cluster. This can occur in heterogeneous clusters, where all EJBs have not been deployed to all WebLogic Server instances. In these cases, WebLogic Server uses a multitier connection to access the datastore, rather than multiple direct connections. This approach uses fewer resources, and yields better performance for the transaction.

However, for best performance, the cluster should be homogeneous — all EJBs should reside on all available WebLogic Server instances.

Indexing with a Version Column

When using an Optimistic concurrency strategy with the `optimistic` column specified as `version`, creating an index on the table consisting of the primary key column and the version column can improve performance.

Tuning WebLogic Server Applications

WebLogic Server only performs as well as the applications running on it. To quote the authors of *Mastering BEA WebLogic Server: Best Practices for Building and Deploying J2EE Applications*: “Good application performance starts with good application design. Overly-complex or poorly-designed applications will perform poorly regardless of the system-level tuning and best practices employed to improve performance.” In other words, a poorly designed application can create unnecessary bottlenecks. For example, resource contention could be a case of bad design, rather than inherent to the application domain.

This section discusses some methods to determine the bottlenecks that can impede your application’s performance:

- “Using Performance Analysis Tools” on page 6-2
- “JDBC Application Tuning” on page 6-2
- “JMS Application Tuning” on page 6-3
- “EJB Application Tuning” on page 6-3
- “Web Services Tuning” on page 6-3
- “Managing Sessions” on page 6-4
- “Using Execute Queues to Control Thread Usage” on page 6-5

Using Performance Analysis Tools

This section is a quick reference for using the OptimizeIt™ and JProbe™ profilers with WebLogic Server.

A profiler is a performance analysis tool that allows you to reveal hot spots in the application that result in either high CPU utilization or high contention for shared resources. For a list of common profilers, see “[Performance Analysis Tools](#)” on page A-6.

Using the JProbe Profiler

The [JProbe Suite](#) is a family of products that provide the capability to detect performance bottlenecks, find and fix memory leaks, perform code coverage, and other metrics. For product details, see <http://www.quest.com/jprobe/>

The JProbe website provides a technical white paper, “[Using Sitraka JProbe and BEA WebLogic Server](#)”, which describes how developers can analyze code with any of the JProbe Suite tools running inside BEA WebLogic Server.

Using the Optimizeit Profiler

The [Optimizeit Profiler](#) from Borland is a performance debugging tool for Solaris and Windows platforms. For product details, see

http://www.borland.com/optimizeit/optimizeit_profiler/index.html.

Borland provides detailed [J2EE Integration Tutorials](#) for the supported versions of Optimizeit Profiler that work with WebLogic Server. For details, see

http://info.borland.com/optimizeit/j2ee_support.html#bea.

JDBC Application Tuning

Most performance gains or losses in a database application are determined by how the application is designed. The number and location of clients, size and structure of DBMS tables and indexes, and the number and types of queries all affect application performance.

For more information on optimizing your applications for JDBC and tuning WebLogic JDBC connection pools, see:

- “[Performance Tuning Your JDBC Application](#)” in *Programming WebLogic JDBC* at <http://e-docs.bea.com/wls/docs81/jdbc/performance.html>

- “[Tuning JDBC Connection Pools](#)” in the WebLogic Server *Administration Console Online Help*

See

http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html

JMS Application Tuning

There are a number of design choices that impact performance of JMS applications. Some others include reliability, scalability, manageability, monitoring, user transactions, message driven bean support, and integration with an application server. In addition, there are WebLogic JMS extensions and features have a direct impact on performance.

For more information on optimizing your applications for JMS and tuning WebLogic JMS, see:

- “[WebLogic JMS Performance Guide](#)” white paper on BEA’s dev2dev Web site

See <http://dev2dev.bea.com/products/wlserver/resources.jsp>

- “[Tuning JMS](#)” in the WebLogic Server *Administration Console Online Help*

See http://e-docs.bea.com/wls/docs81/ConsoleHelp/jms_tuning.html

EJB Application Tuning

“[Tuning WebLogic Server EJBs](#)” describes how to tune WebLogic Server Enterprise Java Beans to match your application needs.

Web Services Tuning

There are some performance issues you should be aware of when you program your WebLogic Web services:

- “[Performance Hints](#)” in *Programming WebLogic Web Services*

See http://e-docs.bea.com/wls/docs81/webserv/trouble.html#perf_hints

- “[Debugging Performance Problems](#)” in *Programming WebLogic Web Services*

See http://e-docs.bea.com/wls/docs81/webserv/trouble.html#debug_perf

Managing Sessions

As a general rule, you should optimize your application so that it does as little work as possible when handling session persistence and sessions. You should also design a session management strategy that suits your environment and application.

Managing Session Persistence

Weblogic Server offers five session persistence mechanisms that cater to the differing requirements of your application. The session persistence mechanisms are configurable at the Web application layer. Which session management strategy you choose for your application depends on real-world factors like HTTP session size, session life cycle, reliability, and session failover requirements. For example, a Web application with no failover requirements could be maintained as a single memory-based session; whereas, a Web application with session fail-over capability could be maintained as replicated sessions or JDBC-based sessions, based on their life cycle and object size.

In terms of pure performance, in-memory session persistence is a better overall choice when compared to JDBC-based persistence for session state. According to the authors of *Session Persistence Performance in BEA WebLogic Server 7.0*: “While all session persistence mechanisms have to deal with the overhead of data serialization and deserialization, the additional overhead of the database interaction impacts the performance of the JDBC-based session persistence and causes it to under-perform compared with the in-memory replication.” However, in-memory-based session persistence requires the use of WebLogic clustering, so it isn’t an option in a single-server environment.

On the other hand, an environment using JDBC-based persistence does not require the use of WebLogic clusters and can maintain the session state for longer periods of time in the database. One way to improve JDBC-based session persistence is to optimize your code so that it has as high a granularity for session state persistence as possible. Other factors that can improve the overall performance of JDBC-based session persistence are: the choice of database, proper database server configuration, JDBC driver, and the JDBC connection pool configuration.

For more information on managing session persistence, see:

- “*Session Persistence Performance in BEA WebLogic Server 7.0*” in the *BEA WebLogic Developers Journal* provides in-depth comparisons of the five session persistence mechanisms supported by WebLogic Server, at <http://www.sys-con.com/weblogic/article.cfm?id=163>

- “[Configuring Session Persistence](http://e-docs.bea.com/wls/docs81/webapp/sessions.html#session-persistence)” in *Assembling and Configuring Web Applications*, at <http://e-docs.bea.com/wls/docs81/webapp/sessions.html#session-persistence>
- “[HTTP Session State Replication](http://e-docs.bea.com/wls/docs81/cluster/failover.html#httpstaterep)” in *Using WebLogic Sever Clusters*, at <http://e-docs.bea.com/wls/docs81/cluster/failover.html#httpstaterep>
- “[Using a Database for Persistent Storage \(JDBC Persistence\)](http://e-docs.bea.com/wls/docs81/webapp/sessions.html#jdbc_persistence)” in *Assembling and Configuring Web Applications*, at http://e-docs.bea.com/wls/docs81/webapp/sessions.html#jdbc_persistence

Minimizing Sessions

Configuring how WebLogic Server manages sessions is a key part of tuning your application for best performance. Consider the following:

- Use of sessions involves a scalability trade-off.
- Use sessions sparingly. In other words, use sessions only for state that cannot realistically be kept on the client or if URL rewriting support is required. For example, keep simple bits of state, such as a user’s name, directly in cookies. You can also write a wrapper class to “get” and “set” these cookies, in order to simplify the work of servlet developers working on the same project.
- Keep frequently used values in local variables.
- Put aggregate objects rather than multiple single objects into the session where possible.

For more information, see “[Setting Up Session Management](http://e-docs.bea.com/wls/docs81/webapp/sessions.html#session-management)” in *Assembling and Configuring Web Applications*, at

<http://e-docs.bea.com/wls/docs81/webapp/sessions.html#session-management>.

Using Execute Queues to Control Thread Usage

You can fine-tune an application’s access to execute threads (and thereby optimize or throttle its performance) by using multiple execute queues in WebLogic Server. However, keep in mind that unused threads represent significant wasted resources in a Weblogic Server system. You may find that available threads in configured execute queues go unused, while tasks in other queues sit idle waiting for threads to become available. In such a situation, the division of threads into multiple queues may yield poorer overall performance than having a single, default execute queue.

Default WebLogic Server installations are configured with a default execute queue, `weblogic.kernel.Default`, which is used by all applications running on the server instance. You may want to configure additional queues to:

- **Optimize the performance of critical applications.** For example, you can assign a single, mission-critical application to a particular execute queue, guaranteeing a fixed number of execute threads. During peak server loads, nonessential applications may compete for threads in the default execute queue, but the mission-critical application has access to the same number of threads at all times.
- **Throttle the performance of nonessential applications.** For an application that can potentially consume large amounts of memory, assigning it to a dedicated execute queue effectively limits the amount of memory it can consume. Although the application can potentially use all threads available in its assigned execute queue, it cannot affect thread usage in any other queue.
- **Remedy deadlocked thread usage.** With certain application designs, deadlocks can occur when all execute threads are currently utilized. For example, consider a servlet that reads messages from a designated JMS queue. If all execute threads in a server are used to process the servlet requests, then no threads are available to deliver messages from the JMS queue. A deadlock condition exists, and no work can progress. Assigning the servlet to a separate execute queue avoids potential deadlocks, because the servlet and JMS queue do not compete for thread resources.

Be sure to monitor each execute queue to ensure proper thread usage in the system as a whole. See [“Tuning the Default Execute Queue Threads” on page 4-6](#) for general information about optimizing the number of threads.

Creating Execute Queues

An execute queue represents a named collection of execute threads that are available to one or more designated servlets, JSPs, EJBs, or RMI objects. An execute queue is represented in the domain `config.xml` file as part of the `Server` element. For example, an execute queue named `CriticalAppQueue` with four execute threads appears in the `config.xml` file as follows:

```
...
<Server
  Name="exampleServer"
  ListenPort="7001"
  NativeIOEnabled="true"/>
<ExecuteQueue Name="default"
  ThreadCount="15"/>
```

```
<ExecuteQueue Name="CriticalAppQueue"
  ThreadCount="4" />
...
</Server>
```

To configure a new execute queue using the Administration Console:

1. Start the Administration Server if it is not already running.
2. Access the Administration Console for the domain.
3. Expand the Servers node in the left pane to display the servers configured in your domain.
4. Right-click the name of the server instance on which you want to add an execute queue, and then select View Execute Queues from the pop-up menu.
5. On the execute queue Configuration tab, click the Configure a New Execute Queue link.
6. On the execute queue Configuration tab, modify the following attributes or accept the system defaults:

- Queue Length: Always leave the Queue Length at the default value of 65536 entries. The Queue Length specifies the maximum number of simultaneous requests that the server can hold in the queue. The default of 65536 requests represents a very large number of requests; outstanding requests in the queue should rarely, if ever reach this maximum value.

If the maximum Queue Length is reached, WebLogic Server automatically doubles the size of the queue to account for the additional work. Note, however, that exceeding 65536 requests in the queue indicates a problem with the threads in the queue, rather than the length of the queue itself; check for stuck threads or an insufficient thread count in the execute queue.

- Queue Length Threshold Percent: The percentage (from 1–99) of the Queue Length size that can be reached before the server indicates an overflow condition for the queue. All actual queue length sizes below the threshold percentage are considered normal; sizes above the threshold percentage indicate an overflow. When an overflow condition is reached, WebLogic Server logs an error message and increases the number of threads in the queue by the value of the Threads Increase attribute to help reduce the workload.

By default, the Queue Length Threshold Percent value is 90 percent. In most situations, you should leave the value at or near 90 percent, to account for any potential condition where additional threads may be needed to handle an unexpected spike in work requests. Keep in mind that Queue Length Threshold Percent must not be used as an

automatic tuning parameter—the threshold should never trigger an increase in thread count under normal operating conditions.

- **Thread Count:** The number of threads assigned to this queue. If you do not need to use more than 15 threads (the default) for your work, do not change the value of this attribute. (For more information, see [“Should You Modify the Default Thread Count?” on page 4-7.](#))
- **Threads Increase:** The number of threads WebLogic Server should add to this execute queue when it detects an overflow condition. If you specify zero threads (the default), the server changes its health state to “warning” in response to an overflow condition in the thread, but it does not allocate additional threads to reduce the workload.

Note: If WebLogic Server increases the number of threads in response to an overflow condition, the additional threads remain in the execute queue until the server is rebooted. Monitor the error log to determine the cause of overflow conditions, and reconfigure the thread count as necessary to prevent similar conditions in the future. Do not use the combination of Threads Increase and Queue Length Threshold Percent as an automatic tuning tool; doing so generally results in the execute queue allocating more threads than necessary and suffering from poor performance due to context switching.

- **Threads Maximum:** The maximum number of threads that this execute queue can have; this value prevents WebLogic Server from creating an overly high thread count in the queue in response to continual overflow conditions. By default, the Threads Maximum is set to 400.
- **Thread Priority:** The priority of the threads associated with this queue. By default, the Thread Priority is set to 5.

7. Click **Create** to create the new execute queue.

8. Reboot the server to use the new settings.

Assigning Servlets and JSPs to Execute Queues

You can assign a servlet or JSP to a configured execute queue by identifying the execute queue name in the initialization parameters. Initialization parameters appear within the `init-param` element of the servlet's or JSP's deployment descriptor file, `web.xml`. To assign an execute queue, enter the queue name as the value of the `wl-dispatch-policy` parameter, as in the example:

```

<servlet>
  <servlet-name>MainServlet</servlet-name>
  <jsp-file>/myapplication/critical.jsp</jsp-file>
  <init-param>
    <param-name>wl-dispatch-policy</param-name>
    <param-value>CriticalAppQueue</param-value>
  </init-param>
</servlet>

```

See “[Initializing a Servlet](#)” in *Programming WebLogic HTTP Servlets* for more information about specifying initialization parameters in `web.xml`.

Assigning EJBs and RMI Objects to Execute Queues

To assign an EJB object to a configured execute queue, use the new `dispatch-policy` element in `weblogic-ejb-jar.xml`. For more information, see the [weblogic-ejb-jar.xml Deployment Descriptor](#), at

http://e-docs.bea.com/wls/docs81/ejb/reference.html#dispatch_policy.

While you can also set the dispatch policy through the appc compiler `-dispatchPolicy` flag, BEA strongly recommends you use the deployment descriptor element instead. This way, if the EJB is recompiled, during deployment for example, the setting will not be lost.

To assign an RMI object to a configured execute queue, use the `-dispatchPolicy` option to the `rmic` compiler. For example:

```
java weblogic.rmic -dispatchPolicy CriticalAppQueue ...
```


Related Reading: Performance Tools and Information

The following sections provide an extensive performance-related reading list:

- [“BEA Systems, Inc. Information” on page A-2](#)
- [“Sun Microsystems Information” on page A-2](#)
- [“Linux OS Information” on page A-3](#)
- [“Hewlett-Packard Company Information” on page A-4](#)
- [“Microsoft Information” on page A-4](#)
- [“Web Performance Tuning Information” on page A-5](#)
- [“Network Performance Tools” on page A-6](#)
- [“Load Testing Tools” on page A-6](#)
- [“Performance Analysis Tools” on page A-6](#)
- [“Production Performance Management” on page A-7](#)
- [“Benchmarking Information” on page A-7](#)
- [“Java Virtual Machine \(JVM\) Information” on page A-8](#)
- [“Enterprise JavaBeans Information” on page A-9](#)
- [“Java Message Service \(JMS\) Information” on page A-9](#)
- [“Java Database Connectivity \(JDBC\) Information” on page A-10](#)

- “General Performance Information” on page A-10

BEA Systems, Inc. Information

- For general information about BEA Systems, see the [BEA Web site](http://www.bea.com)
See <http://www.bea.com>
- [BEA WebLogic Server Documentation page](http://e-docs.bea.com/wls/docs81)
See <http://e-docs.bea.com/wls/docs81>
- [BEA WebLogic Platform Documentation page](http://edocs.bea.com/platform/docs81/index.html)
See <http://edocs.bea.com/platform/docs81/index.html>
- BEA’s [dev2dev](http://dev2dev.bea.com/index.jsp) Web site
See <http://dev2dev.bea.com/index.jsp>
- [BEA WebLogic Server Evaluation White Papers](http://www.bea.com/framework.jsp?CNT=papers.htm&FP=/content/products/server/evaluate/) (for example, “J2EE Design Considerations for WebLogic Server” and “Distributed Computing with BEA WebLogic Server”)
See
<http://www.bea.com/framework.jsp?CNT=papers.htm&FP=/content/products/server/evaluate/>
- *Professional J2EE Programming with BEA WebLogic Server* by Paco Gomez and Peter Zadrozny, 2000
- *BEA WebLogic Server Bible* by Joe Zuffoletto, et al, 2002
- *J2EE Performance Testing with BEA WebLogic Server* by Peter Zadrozny, Philip Aston, and Ted Osborne, 2002
- *Mastering BEA WebLogic Server: Best Practices for Building and Deploying J2EE Applications* by Gregory Nyberg, Robert Patrick, Paul Bauerschmidt, Jeff McDaniel, and Raja Mukherjee, 2003

Sun Microsystems Information

- For general information about Sun Microsystems, see [Sun’s Web site at http://www.sun.com](http://www.sun.com)
- [Sun Microsystems Performance Information](#)

See <http://www.sun.com/sun-on-net/performance.html>

- Java Standard Edition Platform Documentation

See <http://java.sun.com/docs/index.html>

- Java 2 SDK, Standard Edition Documentation

See <http://java.sun.com/j2se/1.4.1/docs>

- *Solaris Tunable Parameters Reference Manual* (Solaris 8)

See <http://docs.sun.com/?p=/doc/816-0607>

- *Solaris 9 Tunable Parameters Reference Manual* (Solaris 9)

See <http://docs.sun.com/?p=/doc/806-7009>

- For BEA WebLogic Server and Solaris-specific details, see the SPARC Solaris links on the **Supported Configurations** pages at

<http://e-docs.bea.com/platform/suppconfigs/index.html>.

- SPARC with Solaris 8
- SPARC with Solaris 9

- For more about Solaris configuration, check the **Solaris FAQ**

See <http://www.science.uva.nl/pub/solaris/solaris2/index.html>

- *Sun Performance and Tuning: Java and the Internet* by Adrian Cockcroft, et al, 1998
- *Solaris 7 Performance Administration Tools* by Frank Cervone, 2000

Linux OS Information

- For general information about the Linux operating system, see **Linux Online** at <http://www.linux.org/>

- For information about the Linux Documentation Project, see **LDP** at <http://www.tldp.org/>

- For information about Redhat Enterprise Linux, see **Redhat** at <http://www.redhat.com/software/rehel/>

- For information about SuSE Linux Enterprise Server, see **SuSE Linux** at <http://www.suse.com/us/business/products/server/sles/index.html>

- [Linux Performance Tuning and Capacity Planning](#), by Jason R. Find, et al, 1997, Sams 2001
- [Ipsysctl Tutorial 1.0.4](#), at <http://ipsysctl-tutorial.frozentux.net/ipsysctl-tutorial.html>, describes the IP options provided by Linux
- [The Linux Cookbook: Tips and Techniques for Everyday Use](#), by Michael Stutz at <http://www.dsl.org/cookbook/>

Hewlett-Packard Company Information

- General [Hewlett-Packard](#) information
See <http://www.thenewhp.com>.
- For BEA WebLogic Server and HP-UX-specific details, see [Hewlett-Packard HP/9000 with HP-UX 11.0 and 11i](#) on the BEA Certifications Pages
See <http://e-docs.bea.com/platform/supconfigs/hp9000.html>
- [Java Performance Tuning on HP-UX](#)
See http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,1602,00.html
- [Hewlett Packard JMeter](#), a tool for analyzing profiling information
See <http://www.hp.com/products1/unix/java/hpjmeter/>
- [GlancePlus](#) system performance diagnostic tool
See <http://www.openview.hp.com/products/gplus/index.html>
- [HPjconfig](#) Java system configuration tool
See <http://www.hp.com/products1/unix/java/java2/hpjconfig/index.html>

Microsoft Information

- General [Microsoft](#) information
See <http://www.microsoft.com>
- [Windows 2000 Performance Tuning](#) White Paper

See

<http://www.microsoft.com/technet/win2000/win2ksrv/technote/perftune.asp>

- *Windows 2000 Performance Guide*, by Mark Friedman and Odysseas Pentakalos, 2002, O'Reilly

See

<http://www.amazon.com/exec/obidos/ASIN/1565924665/qid%3D1055443647/sr%3D11-1/ref%3Dsr%5F11%5F1/104-9412286-0155141>

- [SQL-Server-Performance.Com](http://www.sql-server-performance.com/), Microsoft SQL Server Performance Tuning and Optimization

See <http://www.sql-server-performance.com/>

- *Microsoft SQL Server 2000 Performance Optimization and Tuning Handbook*, by Ken England, 2001, Digital Press

See

http://www.sql-server-performance.com/sql_server_2000_perform_optimization_review.asp

Web Performance Tuning Information

- [Apache Performance Notes](http://httpd.apache.org/docs/misc/perf-tuning.html)

See <http://httpd.apache.org/docs/misc/perf-tuning.html>

- [iPlanet Web Server 4.0 Performance Tuning, Sizing, and Scaling](http://docs.sun.com/db/doc/816-5663-10)

See <http://docs.sun.com/db/doc/816-5663-10>

- *The Art and Science of Web Server Tuning with Internet Information Services 5.0*

See

<http://www.microsoft.com/windows2000/techinfo/administration/web/tuning.asp>

- *Web Performance Tuning: Speeding Up the Web*, by Patrick Killelea, Linda Mui (Editor), O'Reilly Nutshell, 1998
- *Capacity Planning for Web Performance: Metrics, Models, and Methods*, by Daniel A. Menasce, Virgilio A. F. Almeida, Prentice Hall PTR, 1998
- *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*, by Daniel A. Menasce, Virgilio A. F. Almeida, Prentice Hall PTR, 2000

Network Performance Tools

- [TracePlus/Ethernet](#), a network packet analysis tool for Windows 95/98/ME, NT 4.x, Windows 2000/XP

See <http://www.sstinc.com/home.html>

Load Testing Tools

- [LoadRunner](#), a tool that predicts enterprise-level system behavior and performance by emulating thousands of users and employs performance monitors to identify and isolate problems.

See <http://www-heva.mercuryinteractive.com/products/loadrunner/>

- [e-Load](#), a fast and accurate way to perform load testing, scalability testing, stress testing of enterprise Web applications.

See

<http://www.empirix.com/Empirix/Web+Test+Monitoring/testing+solutions/web+application+load+testing.html>

- [The Grinder](#), a pure Java load-testing framework.

See <http://sourceforge.net/projects/grinder/>

Performance Analysis Tools

A profiler is a performance analysis tool that allows you to reveal hot spots in the application that result in either high CPU utilization or high contention for shared resources. Some common profilers are:

- [OptimizeIt Java Performance Profiler](#) from Borland, a performance debugging tool for Solaris and Windows

See http://borland.com/optimizeit/optimizeit_profiler/index.html

- [JProbe Profiler with Memory Debugger](#), a family of products that provide the capability to detect performance bottlenecks, perform code coverage and other metrics

See <http://www.sitraka.com/software/jprobe>

- [Product Review: OptimizeIt vs. JProbe](#), Journal of Object-Oriented Programming, April 2004

See <http://www.adtmag.com/joop/article.asp?id=3668>

- **Hewlett Packard JMeter**, a tool for analyzing profiling information
See <http://www.hp.com/products1/unix/java/hpjmeter/>
- **Topaz, Mercury Interactive's** application performance management solution
See <http://www-svca.mercuryinteractive.com/products/topaz/>
- **VTune Performance Analyzer** a tool to identify and locate performance bottlenecks in your code
See <http://www.intel.com/software/products/vtune/>
- **PerformaSure** a tool to detect, diagnose, and resolve performance problems in multi-tier J2EE applications
See <http://http.java.quest.com/performasure/performasure.shtml>

Production Performance Management

- **Veritas i3 for Web-J2EE** is a monitoring, analysis, and tuning tool for Web-based J2EE Applications.
See <http://www.veritas.com/Products/www?c=product&refId=316>
- **Wily Technology, Inc.** provides management solutions for large-scale, real-time production Web applications, applications servers, portal solutions and integration middleware.
See <http://partners.bea.com/search.portal?partnerId=192>

Benchmarking Information

- **SPECjbb2000**, a software benchmark product developed by the Standard Performance Evaluation Corporation (SPEC). SPECjbb2000 is designed to measure a system's ability to run Java server applications.
See <http://www.spec.org/osg/jbb2000/docs/whitepaper.html>
- **ECPerf Benchmark Kit**, a software benchmark product developed under the Java Community ProcessSM Program that is designed to measure performance and scalability and assist the J2EE user in understanding J2EE scalability and tuning.
See <http://developer.java.sun.com/developer/releases/j2ee/ecperf>

- [SPECjAppServer2001](#) (Java Application Server), a client/server benchmark for measuring the performance of Java Enterprise Application Servers using a subset of J2EE API's in a complete end-to-end web application.

See <http://www.spec.org/osg/jAppServer2001>

Java Virtual Machine (JVM) Information

- [WebLogic JRockit Documentation](#), at http://e-docs.bea.com/more_jrockit.html
- [JVM Corner at artima.com](#)

See <http://www.artima.com/java/index.html>

- [Sun Microsystems FAQ](#) about Java HotSpot technology and about performance in general

See <http://java.sun.com/docs/hotspot/PerformanceFAQ.html>

- [Performance Documentation for the Java HotSpot Virtual Machine](#)

See <http://java.sun.com/docs/hotspot/index.html>

- [Java HotSpot VM Options](#), a Sun Microsystems document provides information on the command-line options and environment variables that can affect the performance characteristics of the HotSpot JVM.

See <http://java.sun.com/docs/hotspot/VMOptions.html>

- [Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1](#), a Sun Microsystem document on how to reduce garbage collection times with JDK 1.4.1.

See <http://wireless.java.sun.com/midp/articles/garbagecollection2/>

- [The Java Virtual Machines for J2SE 1.4](#)

See <http://java.sun.com/j2se/1.4.1/docs/guide/vm/index.html>

- [Which Java VM scales best?](#) From JavaWorld, results of a VolanoMark 2.0 server benchmark show how 12 virtual machines stack up.

See <http://www.javaworld.com/jw-08-1998/jw-08-volanomark.html>

- *Garbage Collection: Algorithms for Automatic Dynamic Memory Management* by Richard Jones, Rafael D Lins, John Wiley & Sons, 1999

See

<http://www.amazon.com/exec/obidos/ASIN/0471941484/richardjones/002-1748120-9756040>

Enterprise JavaBeans Information

- *Programming WebLogic Enterprise JavaBeans*

See <http://e-docs.bea.com/wls/docs81/ejb/index.html>

- *Enterprise JavaBeans, Second Edition*, by Richard Monson-Haefel, Mike Loukides (Editor), 2000

- *Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition*, by Ed Roman, 1999

- [TheServerSide.com](http://www.theserverside.com), a free online community dedicated to Enterprise JavaBeans (EJBs) and J2EE.

See <http://www.theserverside.com/home/index.jsp>

- *Seven Rules for Optimizing Entity Beans*, by Akara Sucharitakul, Java Developer Connection, 2001

See

<http://developer.java.sun.com/developer/technicalArticles/ebeans/sevenrules/>

Java Message Service (JMS) Information

- *Programming WebLogic JMS*

See <http://e-docs.bea.com/wls/docs81/jms/index.html>

- “Tuning JMS” in the WebLogic Server *Administration Console Online Help*

See http://e-docs.bea.com/wls/docs81/ConsoleHelp/jms_tuning.html

- “WebLogic Messaging Bridge” in the WebLogic Server *Administration Console Online Help*

See http://e-docs.bea.com/wls/docs81/ConsoleHelp/messaging_bridge.html

- *WebLogic JMS Performance Guide* white paper on the BEA dev2dev Web site

See <http://dev2dev.bea.com/products/wlserver/resources.jsp>

- [JMS Specification](#)

See <http://java.sun.com/products/jms/docs.html>

Java Database Connectivity (JDBC) Information

- *Performance Tuning Your JDBC Application*

See <http://e-docs.bea.com/wls/docs81/jdbc/performance.html>

- “Increasing Performance with the Statement Cache” in the WebLogic Server *Administration Console Online Help*

See

http://e-docs.bea.com/wls/docs81/ConsoleHelp/jdbc_connection_pools.html#statementcache

General Performance Information

- [Jack Shirazi's Java Performance Tuning](#) Web site

See <http://www.javaperformancetuning.com>

- “Avoiding Scalability Shock” by Bill Shea, May/June 2000

See <http://www.stickyminds.com> and search for “Avoiding Scalability Shock” or “Bill Shea”.

- *Performance and Idiom Guide* by Craig Larman and Rhett Guthrie, 1999

Index

A

- AcceptBacklog attribute 4-14
- Activation, stateful session EJBs 5-5

B

- Bandwidth, network 2-8
- Benchmarking, related reading A-7
- bulk insert support 5-12
- Bull IBM
 - hardware tuning 2-2

C

- client option, Solaris HotSpot VM 3-11
- client option, Windows HotSpot VM 3-11
- cluster
 - distributing transactions across EJBS 5-12
- Clusters, scalability 4-18
- Command-line options, Java
 - Linux 3-11, 3-12
 - Solaris 3-11, 3-12
 - Windows 3-11, 3-12
 - Windows, non-standard 3-12
- compileCommand parameter, jsp-descriptor element 4-17
- Compilers
 - changing in Console 4-16
 - changing in weblogic.xml 4-17
 - setting a 4-16
- config.xml parameters, tuning 4-2
- Connection backlog buffering 4-14
- Connection pool size, JDBC 4-15

- Connection pools, database 4-15
- Container classes, compiling EJB 4-17
- Customer support contact information xv

D

- Database connection pools 4-15
- database insert
 - bulk insert 5-12
- Documentation, where to find it xiv
- Domain, WebLogic Server 4-21

E

- Eden/survivor space, setting heap ratios 3-8
- EJB
 - activation 5-4
 - caching size 5-4
 - container classes, compiling 4-17
 - lifecycle of stateless session EJBs 5-3
 - parameters, tuning 5-1, 5-6
 - passivation 5-4
 - pool size, setting 5-2
 - related reading A-9
- EJB support
 - bulk insert 5-12

F

- Forcing garbage collection 3-10

G

- Garbage collection

- forcing on a server 3-10
- tuning 3-3
 - tuning, 1.3.1 JVM A-8
- General performance, related reading A-10
- Green threads vs. native threads 3-2

H

- Hardware tuning 2-1
 - Bull IBM 2-2
 - Hewlett-Packard 2-2, 2-3
 - Intel Pentium 2-2
 - network 2-8
 - platform-specific 2-1
 - Solaris 2-3
- Heap size
 - setting maximum 3-8
 - setting minimum 3-8
 - specifying values 3-6
 - tuning 3-3
- Heap size ratios 3-8
- Hewlett-Packard
 - hardware tuning 2-2, 2-3
 - related reading A-4
- hotspot option
 - Linux HotSpot Client VM 3-11
 - Linux HotSpot Server VM 3-11
 - Solaris HotSpot Client VM 3-11
 - Solaris HotSpot Server VM 3-11
 - Windows HotSpot Client VM 3-11
 - Windows HotSpot Server VM 3-11
- hotspot option, Linux HotSpot Client VM 3-11

I

- In-memory replication 6-4
- Intel Pentium
 - hardware tuning 2-2
- Isolation level, setting transaction 5-6
- isolation-level element 5-6

J

- Java command-line options
 - Linux 3-11, 3-12
 - Solaris 3-11, 3-12
 - Windows 3-11, 3-12
 - Windows, non-standard 3-12
- Java compiler, setting 4-16
- JDBC application tuning 6-2
- JDBC connection pool size 4-15
- JDBC-based persistence 6-4
- JMeter, Hewlett Packard profiler A-4, A-7
- JMS application tuning 6-3
- JMS, related reading A-9, A-10
- JProbe profiler 6-2, A-6
 - related reading A-6
- jsp-descriptor element, weblogic.xml 4-17
- JSPs, precompiling 4-17
- JVMs
 - mixed client/server 3-2
 - related reading A-8
 - verbosegc option 3-5

L

- LAN infrastructure 2-9
- Linux
 - java command-line options 3-11, 3-12

M

- max-beans-in-cache element 5-4
- max-beans-in-free-pool element 5-2
- Maximum heap size, setting 3-8
- Maximum memory, operating system tuning 2-8
- Maximum New generation heap size, setting 3-8
- MaxNewSize option 3-8
- Microsoft, related reading A-4
- Minimizing sessions 6-5
- Minimum heap size, setting 3-8
- Mixed client/server JVMs 3-2

N

- Native threads vs. green threads 3-2
- NativeIOEnabled attribute 4-5
- Network tuning
 - bandwidth 2-8
 - hardware and software 2-8
 - LAN infrastructure 2-9
 - performance tools A-6
- New generation heap size, setting 3-8
- NewSize option 3-8

O

- Operating system tuning
 - max memory for user process 2-8
- OptimizeIt Profiler
 - related reading A-6
 - using 6-2

P

- Passivation, stateful session EJBs 5-5
- Performance analysis tools
 - related reading A-6
 - using JProbe and OptimizeIt 6-2
- Performance packs
 - enabling via Console 4-5
 - using 4-5
 - which platforms? 4-5
- Persistence
 - JDBC-based 6-4
 - session, managing 6-4
- Platform-specific
 - hardware tuning 2-1
 - JVM tuning 3-2
- Pool size, database connection 4-15
- Precompiling JSPs 4-17
- Printing product documentation xiv
- Profilers
 - related reading A-6
 - using 6-2

R

- Ratios, setting heap size 3-8
- Related reading A-1
 - BEA Systems A-2
 - benchmarking A-7
 - EJBs A-9
 - general performance A-10
 - Hewlett Packard A-4
 - JMS A-9, A-10
 - JVMs A-8
 - Microsoft A-4
 - network performance tools A-6
 - performance analysis tools A-6
 - profilers A-6
 - Sun Microsystems A-2
- Replication, in-memory 6-4

S

- Scalability, clusters 4-18
 - server option, Linux HotSpot VM 3-11
 - server option, Solaris HotSpot VM 3-11
 - server option, Windows HotSpot VM 3-11
- Session management 6-5
- Session persistence
 - in-memory replication 6-4
 - managing 6-4
- Setting Java HotSpot VM options 3-10
- Socket readers, allocating threads 4-9
- Solaris
 - hardware tuning 2-3
 - java command-line options 3-11, 3-12
- SPECjbb2000 A-7
- Standardized benchmarks and metrics 2-1
- Start-up scripts for Administration Server 3-7, 4-1
 - startWebLogic.cmd
 - heap size values 3-7
 - startWebLogic.sh
 - heap size values 3-7
- Stateful session EJBs

- activation and passivation 5-5
- stateless session
 - lifecycle of these EJBs 5-3
- Sun Microsystems, related reading A-2
- Support, technical xv
- SurvivorRatio option 3-8

T

- TCP connections 4-14
- Thread count
 - modifying 4-7
 - scenarios 4-8
 - setting 4-6
 - too high 4-8
 - too low 4-8
- ThreadCount attribute 4-6
- Threading models, UNIX 3-2
- ThreadPoolPercentSocketReaders attribute 4-9
- Threads, socket reader 4-9
- TracePlus/Ethernet A-6
- Transaction isolation level, setting 5-6
- transactions
 - distributing across EJBS in a cluster 5-12
- Tuning
 - config.xml parameters 4-2
- Tuning weblogic-ejb-jar.xml parameters 5-1, 5-6

U

- UNIX threading models 3-2
- Using profilers 6-2

V

- verbosegc option
 - JVM 3-5

W

- WebLogic Server
 - clusters 4-18

- free pool 5-3
- monitoring a domain 4-21
- performance packs 4-5
- tuning 4-1, 5-1
- weblogic.ejb utility 4-17
- weblogic-ejb-jar.xml parameters, tuning 5-1, 5-6
- Windows
 - java command-line options 3-11, 3-12
 - java command-line options non-standard 3-12

X

- Xms option 3-8
- XX
 - MaxNewSize option 3-8
 - NewSize option 3-8
 - SurvivorRatio option 3-8