BEA WebLogic Server™

Performance and Tuning

BEA WebLogic Server Version 6.1
Document Date: June 24, 2002

BEA WebLogic Server Performance and Tuning

Part Number	Document Date	Software Version
N/A	June 24, 2002	BEA WebLogic Server Version 6.1

Contents

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, “Tuning Hardware, Operating System, and Network Performance,” discusses hardware, operating system, and network performance issues.
- Chapter 2, “Tuning Java Virtual Machines (JVMs),” discusses JVM tuning considerations.
- Chapter 3, “Tuning WebLogic Server,” contains information on how to tune WebLogic Server to match your application needs.
- Chapter 4, “Tuning WebLogic Server Applications,” discusses application tuning considerations.
- Appendix A, “Related Reading,” provides an extensive performance-related reading list.

The document also contains an index.

What You Need to Know

This document is intended for people involved with tuning the components in a WebLogic Server platform. It is assumed that readers know server administration and performance tuning fundamentals, the WebLogic Server platform, XML, and Java programming.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site at `http://www.bea.com`. From the BEA Home page, click on Product Documentation.

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 at `http://www.adobe.com`.

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, 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.
italics	Emphasis and book titles.

`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.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

`monospace italic text`	Variables in code. *Example*: `String` *`CustomerName`*`;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.

`[ ]`	Optional items in a syntax line. *Example*:
	`java utils.MulticastTest -n` *`name`* `-a` *`address`*
	     `[-p` *`portnumber`*`] [-t` *`timeout`*`] [-s` *`send`*`]`

Convention	Usage

|	Separates mutually exclusive choices in a syntax line. *Example*:
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`     password {application} {source}`

`...`	Indicates one of the following in a command line: ■ 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.

CHAPTER

1 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 1-1
- "Operating System Tuning" on page 1-3
- "Network Performance" on page 1-8

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, you may want to review the Standard Performance Evaluation Corporation, at `www.spec.org`, which provides a set of standardized benchmarks and metrics for evaluating computer system performance.

Supported Platforms

The following table provides selected links to the information on the Supported Configurations pages, at `http://e-docs.bea.com/wls/certifications/certs_610/index.html`, which contains a complete listing of the latest certification information on the hardware/operating system platforms that are supported for each release of WebLogic Server.

Table 1-1 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/wls/certifications/certs_610/index.html`. ■ Bull/IBM pSeries with AIX 5L v5.1 ■ Bull/IBM pSeries with AIX 5L v5.2
Hewlett-Packard 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/wls/certifications/certs_610/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/wls/certifications/certs_610/index.html`. ■ Windows 2000 Server or Windows 2000 Advanced Server ■ Windows 2000 Professional ■ Windows NT ■ Windows XP See also "Microsoft Information" on page A-4.

Table 1-1 Platform-Specific Tuning Information

Platform	For more information
Intel 32-bit-compatible with Red Hat Linux and SuSE Linux	See the Red Hat Linux and SuSE Linux links on the Supported Configurations pages at `http://e-docs.bea.com/wls/certifications/certs_610/index.html`. ■ SuSE Linux (SLES 7) for IA-32 See also “Linux OS Information” on page A-3.
IBM S/390 and Z-Series compatible with Red Hat Linux and SuSE Linux	See the Red Hat Linux and SuSE Linux links on the Supported Configurations pages at `http://e-docs.bea.com/wls/certifications/certs_610/index.html`. ■ Red Hat Linux for IBM S/390 and Z-Series ■ SuSE Linux (SLES 7) for IBM S/390 and Z-Series ■ SuSE Linux (SLES 8) for IBM S/390 and Z-Series See also “Linux OS Information” on page A-3.
Sun Microsystems SPARC with Solaris	See the Sun Microsystems SPARC Solaris links on the Supported Configurations pages at `http://e-docs.bea.com/wls/certifications/certs_610/index.html`. ■ SPARC with Solaris 2.6 ■ SPARC with Solaris 2.7 ■ SPARC with Solaris 8 ■ SPARC with Solaris 9 See also “Sun Microsystems Information” on page A-2.

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/wls/certifications/certs_610/index.html`.

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

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 1-2 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/tcp tcp_naglim_def`	`1`
`/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 1-3 Suggested /etc/system Values

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

CE Gigabit Network Card Settings

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

Table 1-4 Suggested CE Gigabit Card Values

Parameter	Suggested Value
`set ce:ce_bcopy_thresh`	`256`
`set ce:ce_dvma_thresh`	`256`
`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 1-5 Suggested Linux Values

Parameter	Suggested Value
`/sbin/ifconfig lo mtu`	`1500`
`kernel.msgmni`	`1024`
`kernel.sem`	`1000 32000 32 512`
`fs.file-max`	`65535`

Table 1-5 Suggested Linux Values

Parameter	Suggested Value
`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.

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/communications/networkbasics/tcpip_implement.asp`.
- For HP-UX tuning information, see the *Tunable Kernel Parameters* reference documentation, at `http://docs.hp.com/hpux/onlinedocs/TKP-90203/TKP-90203.html`.
- 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 2, "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 1-4.

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.

2 Tuning Java Virtual Machines (JVMs)

The Java virtual machine (JVM) is an "execution engine" that executes the byte codes in Java class files on a microprocessor. How you tune your JVM affects the performance of WebLogic Server and your applications.

This document discusses the following JVM tuning topics:

For links to related reading, see "Java Virtual Machine (JVM) Information" on page A-7.

JVM Tuning Considerations

Table 2-1 presents general JVM tuning considerations.

Table 2-1 General JVM Tuning Considerations

Issue	Description
JVM vendor and version	Use only production JVMs on which WebLogic Server has been certified. WebLogic Server 6.x supports only those JVMs that are Java 1.3-compliant. The Platform Support page at `http://e-docs.bea.com/wls/platforms/index.html` is frequently updated and contains the latest certification information on various platforms.
Tuning heap size and garbage collection	For tuning details, see "About JVM Heap Size" on page 2-3. For a good overview of garbage collection, see Tuning Garbage Collection with the 1.3.1 Java Virtual Machine, at `http://java.sun.com/docs/hotspot/gc/`.
Generational garbage collection	See "About Generational Garbage Collection" on page 2-4.
Mixed client/server JVMs	Deployments using different JVM versions for the client and server are supported in WebLogic Server. For information about support for mixed client/server JVMs, see `http://e-docs.bea.com/wls/platforms/index.html#mix`.
UNIX threading models	There are two UNIX threading models: green threads and native threads. To get the best performance and scalability with WebLogic Server, choose a JVM that uses native threads. For Solaris, see "Threading Models and Solaris Versions Supported" on the JavaSoft web site at `http://www.javasoft.com/products/jdk/1.1/solaris-product-comparison.html#threading`.

Table 2-1 General JVM Tuning Considerations (Continued)

Issue	Description
Just-in-Time (JIT) JVMs	Use a JIT compiler when you run WebLogic Server. Most JVMs use a JIT compiler, including those from Sun Microsystems and Symantec. See your JVM supplier documentation for more information. **Note:** *Sun's JVM 1.3.x, JIT options are no longer valid. See "Java Virtual Machine (JVM) Information" on page A-7.*

About JVM Heap Size

Garbage collection is the VM process of de-allocating 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, the object is garbage.

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 you spend doing garbage collection while maximizing the number of clients that you 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 2-7.

About Generational Garbage Collection

The 1.3 Java HotSpot JVM uses generational garbage collection. While naive garbage collection examines every living object in the heap, generational garbage collection considers the lifetime of an object to avoid extra work.

The heap is divided into two general areas: New and Old. The New generation area is sub-divided further into Eden and two survivor spaces. Eden is the area where new objects are allocated. When garbage collection occurs, live objects in Eden are copied into the next survivor space. Objects are copied between survivor spaces in this way until they exceed a maximum threshold, and then they are moved out of the New area and into the Old. For information about specifying the size and ratios of the New and Old generation areas, see "Specifying Heap Size Values" on page 2-7.

Many objects become garbage shortly after being allocated. These objects are said to have "infant mortality." The longer an object survives, the more garbage collection it goes through, and the slower garbage collection becomes. The rate at which your application creates and releases objects determines how often garbage collection occurs. Attempt to cache objects for re-use, whenever possible, rather than creating new objects.

Knowing that a majority of objects die young allows you to tune for efficient garbage collection. When you manage memory in generations, you create memory pools to hold objects of different ages. Garbage collection can occur in each generation when it fills up. If you can arrange for most of your objects to survive less than one collection, garbage collection is very efficient. Poorly sized generations cause frequent garbage collection, impacting your performance.

For a good overview of generational garbage collection, see Tuning Garbage Collection with the 1.3.1 Java Virtual Machine, at `http://java.sun.com/docs/hotspot/gc/`.

Determining Heap Size

This section describes basic steps for determining the most effective heap size:

1. Monitor the performance of WebLogic Server under maximum load while running your application.
2. Use the `-verbosegc` option to measure exactly how much time and resources are put into garbage collection.
3. Turn on verbose garbage collection output for your Java VM and redirect *both* the standard error and standard output to a log file.

   See "Turning On Verbose Garbage Collection and Redirecting Output" on page 2-6.
4. Analyze the following:
   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% free, you might set the heap size smaller.
5. If you are using 1.3 Java HotSpot JVM, set generation sizes.

   "Specifying Heap Size Values" on page 2-7.
6. 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.

7. 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% of the available RAM (not taken by the operating system or other processes) for your JVM.

8. If you find that you have a large amount of RAM remaining, run more WebLogic Servers on your machine.

See also “Specifying Heap Size Values” on page 2-7

Turning On Verbose Garbage Collection and Redirecting Output

This section describes how to turn on verbose garbage collection and redirect output to a log file for diagnostic purposes:

1. Turn on verbose garbage collection output for your Java VM when you start WebLogic Server, as shown in the example in the following step.

2. 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:

   ```
   % java -ms64m -mx64m -verbosegc -classpath $CLASSPATH
   -Dweblogic.domain=mydomain -Dweblogic.Name=clusterServer1
   -Djava.security.policy==/bea/weblogic6x/lib/weblogic.policy
   -Dweblogic.management.server=192.168.0.101:7001
   -Dweblogic.management.username=system
   -Dweblogic.management.password=systemPassword weblogic.Server
   >> logfile.txt
   ```

On HPUX, use the following option to redirect `stderr stdout` to a single file:

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

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

On Solaris, use the following command:

```
weblogic.Server > server.out 2>&1
```

Specifying Heap Size Values

You can specify Java heap size values when starting the WebLogic Administration Server from the Java command line. For example:

```
$ 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.

Be aware that WebLogic Server startup and environment scripts affect these parameters. Sample scripts are provided with the WebLogic distribution for starting the default server and for setting the environment to build and run the server:

- `startWebLogic.cmd` and `setEnv.cmd` for Windows systems
- `startWebLogic.sh` and `setEnv.sh` for UNIX systems.

You will need to modify these scripts to fit your environment and applications. See Starting and Stopping WebLogic Servers at `http://e-docs.bea.com/wls/docs61/adminguide/startstop.html`.

Java Heap Size Options

You achieve best performance by individually tuning each of your applications. Configuring the JVM heap size options listed in Table 2-2 increases performance for most applications.

The options listed in Table 2-2 may differ depending on your architecture and operating system. See your vendor's documentation for platform-specific JVM tuning options.

Table 2-2 Java Heap Size Options

Task	Option	Description
Setting the New generation heap size	`-XX:NewSize`	Use this option to set the New generation Java heap size. 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. 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.
Setting the maximum New generation heap size	`-XX:MaxNewSize`	Use this option to set the maximum New generation Java heap size. Set this value to a multiple of 1024 that is greater than 1MB.
Setting New heap size ratios	`-XX:SurvivorRatio`	The New generation area is divided into three sub-areas: Eden, and two survivor spaces that are equal in size. Use the `-XX:SurvivorRatio=X` option to configure the ratio of the Eden/survivor space size. Try setting this value to 8 and then monitor your garbage collection.
Setting minimum heap size	`-Xms`	Use this option to 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`).
Setting maximum heap size	`-Xmx`	Use this option to set the maximum Java heap size. Set this value to a multiple of 1024 that is greater than 1MB.

Forcing Garbage Collection

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

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

1. On the Administration Console, click the server instance node in the left pane for the server whose memory usage you want to view. A dialog displays in the right pane showing the tabs associated with this instance.
2. Click the Monitoring tab.
3. Click the JVM tab.
4. Check the Memory Usage graph for high usage.

   Note that the Memory Usage graph displays only for servers that are currently running.
5. Click the Force Garbage Collection text link to force garbage collection.

   A message displays indicating that the collection operation was successful.

Setting Java HotSpot VM Options

You can use standard `java` options to improve performance. Be aware that 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.

You need to test both your client and server JVMs and see what performs better for your particular application.

See "Setting Non-Standard Java Command Line Options" on page 2-11 for more VM options that affect performance.

Standard Options for NT

For NT, WebLogic Server invokes the JVM via the `java` command. Use the options listed in listed in Table 2-3.

Table 2-3 Standard options for HotSpot VM on NT

Option	Description
`-hotspot`	Selects the Client HotSpot VM.
`-server`	Selects the server VM.
`-classic`	Selects the classic VM.

Standard Options for UNIX

For UNIX, the WebLogic Server invokes the JVM via the `java` command. Use the options listed in listed in Table 2-4.

Table 2-4 Standard options for HotSpot VM on UNIX

Option	Description
`-client or -hotspot`	Selects the Client HotSpot VM.
`-server`	Selects the server VM.

Setting Non-Standard Java Command Line Options

You can use non-standard `java` options to improve performance. Be aware that 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.

Non-Standard Options for NT

Some examples of non-standard options for improving performance on the Hotspot VM on NT are listed in Table 2-5.

Table 2-5 Non-standard options for HotSpot VM on NT

Option	Description
`-Xnoclassgc`	This option disables garbage collection for the class, It prevents re-loading of the class when the class is referenced after all references to it have been lost. This option requires a greater heap size.
`-oss`	This option controls the Java thread stack size. Setting it too high (>2MB) severely degrades performance.
`-ss`	This option controls the native thread stack size. Setting it too high (>2MB) severely degrades performance.

Table 2-5 Non-standard options for HotSpot VM on NT (Continued)

Option	Description
`-Xverbosegc:file=/tmp/gc$$.out`	This option redirects `-verbosegc` messages to a file, allowing you to separate your garbage collection messages from the rest of the messages on `stderr`. It also provides a performance advantage because writes to files are buffered better than writes to a character stream like `stderr`. See also "Turning On Verbose Garbage Collection and Redirecting Output" on page 2-6.

Non-Standard Options for Solaris

See non-standard options for Solaris VMs at `http://java.sun.com/j2se/1.3/docs/tooldocs/solaris/java.html#nonstandard`.

CHAPTER

3 Tuning WebLogic Server

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

- "Tuning config.xml File Parameters" on page 3-1
- "Tuning weblogic-ejb-jar.xml Parameters" on page 3-9
- "Tuning Parameters for Starting WebLogic Server" on page 3-14
- "Setting Your Java Compiler" on page 3-14
- "WebLogic Server Clusters and Scalability" on page 3-16
- "Monitoring a WebLogic Server Domain" on page 3-18

Tuning config.xml File Parameters

Table 3-1 lists the `config.xml` file parameters that impact server performance:

Table 3-1 Performance-Related config.xml Elements

Element	Attribute	For more information
`Server`	`NativeIOEnabled`	See "Using WebLogic Server Performance Packs" on page 3-2.

Table 3-1 Performance-Related config.xml Elements (Continued)

Element	Attribute	For more information
`ExecuteQueue`	`ThreadCount`	See "Setting Thread Count" on page 3-3.
`Server`	`ThreadPoolPercentSocketReaders`	See "Allocating Threads to Act as Socket Readers" on page 3-6.
`Server`	`AcceptBacklog`	See "Tuning Connection Backlog Buffering" on page 3-8.
`JDBCConnectionPool`	`InitialCapacity` `MaxCapacity`	See "Tuning JDBC Connection Pool Size" on page 3-7.

Using WebLogic Server Performance Packs

Benchmarks show major performance improvements in WebLogic Server when you use the performance pack for your platform. Performance packs use a platform-optimized (native) socket multiplexor to improve server performance.

To use a performance pack, make sure the `NativeIOEnabled` attribute of the `Server` element is defined in your `config.xml` file. The default `config.xml` file shipped with your distribution enables this attribute by default: `NativeIOEnabled=true`.

Which Platforms Have Performance Packs?

To see which platforms currently have performance packs available:

1. Go to the Platform Support page at `http://e-docs.bea.com/wls/platforms/index.html`.
2. Choose Edit→Find to locate all instances of "performance pack included".

Enabling Performance Packs

To use the Administration Console to make sure performance packs are enabled:

1. Start the WebLogic Server Console.
2. Open the Servers folder in the navigation tree.
3. Select your server (`myserver` in a default installation) in the Servers folder.
4. Select the Configuration tab.
5. Select the Tuning tab.
6. If the Native IO Enabled check box is not selected, select the check box.
7. Click Apply.
8. Restart your sever.

Setting Thread Count

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 a WebLogic Server instance, 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 default execute queue (named “default”) that contains 15 threads. WebLogic Server instances also contain two built-in execute queues named `__weblogic_admin_html_queue` and `__weblogic_admin_rmi_queue`, but these queues are reserved for communicating with the WebLogic Administration Console. If you configure no additional execute queues, all Web applications and RMI objects use the default queue.

Note: For most applications, leave the default value unchanged.

Modifying 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. You can degrade performance by increasing the value of the

`ThreadCount` attribute unnecessarily. Threads consume memory. 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. Suppose your client application is thin and does a lot of its work through remote invocation. The time your thin client application spends connected will be greater than for a client application that does a lot of client-side processing.

If you do not need to use additional threads for your work, do not change the value of this attribute. The thread will not be held for the client application.

If your application makes database calls that take a long time to return, you need more execute threads than an application that makes calls that are short and turn over very rapidly. For the latter, you can use a small number of execute threads and improve performance.

Thread Count Scenarios

Table 3-2 shows some possible scenarios adjusting available threads in the default execute queue. These scenarios assume that all thread requests are satisfied by using the default execute queue. If you configure additional execute queues and assign applications to specific queues, results must be monitored on a pool-by-pool basis.

Table 3-2 Thread Count Scenarios

When...	Results	Do This:
Thread Count < number of CPUs	Results in an under-utilized CPU.	Increase the thread count.
(Thread Count == number of CPUs)	Theoretically ideal, but the CPUs are under-utilized.	Increase the thread count.
(Thread Count > number of CPUs) by a moderate number of threads	Practically ideal, resulting in a moderate amount of context switching and a high CPU utilization rate.	Tune the moderate number of threads and compare performance results.

Table 3-2 Thread Count Scenarios (Continued)

When...	Results	Do This:
(Thread Count > number of CPUs) by many threads	Results in too much context switching which could lead to significant performance degradation.	Reduce the number of threads. For example, if you have 4 processors, then 4 threads can be running concurrently. So, you want the execute threads to be 4 + (the number of blocked threads). This is very application-dependent. For instance, the length of time the application might block threads can invalidate the formula.

Symptoms: Thread Count Too Low

If your thread count is too low, these symptoms appear when WebLogic Server is running under maximum load:

- The CPU is waiting to do work, but there is work that could be done.
- The CPU never reaches 100% utilization.

Symptoms: Thread Count Too High

If your thread count is set too high when you run WebLogic Server under maximum load, your performance increases as you decrease the number of threads.

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 4-5 for more information on assigning applications to configured execute queues.

Allocating Threads to Act as Socket Readers

To set the maximum percentage of execute threads that read messages from a socket, use the `ThreadPoolPercentSocketReaders` attribute in the `config.xml` file. The optimal value for this attribute is application-specific. The default is value 33, and the valid range is 1-99.

The `ThreadPoolPercentSocketReaders` attribute sets the maximum percentage of execute threads that are set to read messages from a socket. 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.

When the native performance packs are not being used, execute threads must be allocated to act as socket reader threads. If possible, use the Performance Pack for your platform. See “Using WebLogic Server Performance Packs” on page 3-2.

Note: Due to a known problem in WebLogic Server 6.1, if you are not using the Performance Pack for your platform, and are allocating threads to act as socket readers, you must set the desired thread pool size as a command line option when starting a Managed Server. For more information and instructions, see “CR179419” in *WebLogic Server 6.1 Release Notes*.

How 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 this 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 users.

Tuning JDBC Connection Pool Size

The `InitialCapacity` attribute of the `JDBCConnectionPool` element allows you to set the number of physical database connections to create when configuring the pool. If the server is unable to create this number of connections, the creation of this connection pool will fail. See "Tuning JDBC Connection Pool Initial Capacity" on page 3-7.

The `MaxCapacity` attribute of the `JDBCConnectionPool` element allows you to set the maximum number of physical database connections in a connection pool. See "Tuning JDBC Connection Pool Maximum Capacity" on page 3-8.

For additional JDBC tuning information, see "Performance Tuning Your JDBC Application" in *Programming WebLogic JDBC*, at `http://e-docs.bea.com/wls/docs61/jdbc/performance.html`.

Tuning JDBC Connection Pool Initial Capacity

During development, it is convenient to set the value of the `InitialCapacity` attribute to a low number. This helps the server start up faster.

In production systems, consider setting `InitialCapacity` equal to the `MaxCapacity` so that all database connections are acquired during server start-up.

If `InitialCapacity` is less than `MaxCapacity`, the server then 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.

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.

Tuning Connection Backlog Buffering

The `AcceptBacklog` attribute of the `Server` element in the `config.xml` file allows you to set the number of connection requests the server will accept before refusing additional requests. The `AcceptBacklog` attribute specifies how many 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 dependant.

Select `Server`→`Configuration`→`Tuning` from the Administration Console to enter a value for the `Accept Backlog` attribute.

During operations, if many connections are dropped or refused at the client, and there are no other error messages on the server, the `AcceptBacklog` attribute might be set too low.

If you are getting “connection refused” messages when you try to access WebLogic Server, raise the value of the `AcceptBacklog` attribute from the default by 25%. Continue increasing the attribute’s value by 25% until the messages cease to appear.

Tuning weblogic-ejb-jar.xml Parameters

Table 3-3 lists the `weblogic-ejb-jar.xml` file parameters that impact performance.

Table 3-3 Performance-Related weblogic-ejb-jar.xml Parameters

Element	For more information
`max-beans-in-free-pool`	See "Setting EJB Pool Size" on page 3-9.
`initial-beans-in-free-pool`	See "Tuning Initial Beans in Free Pool" on page 3-11.
`max-beans-in-cache`	See "Setting EJB Caching Size" on page 3-11.
`concurrency-strategy`	See "Deferring Database Locking" on page 3-13.
`isolation-level`	See "Setting Transaction Isolation Level" on page 3-13.

The following sections describe these elements.

Setting EJB Pool Size

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. By default, `max-beans-in-free-pool` has no limit; the maximum number of beans in the free pool is limited only by the available memory.

This section discusses the following topics:

- "Allocating Pool Size for Session and Message Beans" on page 3-10
- "Allocating Pool Size for Entity Beans" on page 3-10
- "Tuning the Pool Size" on page 3-11

See also:

- max-beans-in-free-pool in *Programming WebLogic Enterprise JavaBeans* at `http://e-docs.bea.com/wls/docs61/ejb/reference.html#max_beans_in_free_pool_60`
- "Using max-beans-in-free-pool" in *Programming WebLogic Enterprise JavaBeans* at `http://e-docs.bea.com/wls/docs61/ejb/EJB_environment.html#Using_max_beans_in_free_pool`

Allocating Pool Size for Session and Message Beans

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. The only reason to change the setting would be to limit the number of beans running in parallel.

Allocating Pool Size for Entity Beans

There is a pool of anonymous entity beans that are used for invoking finders, home methods, and creation of entity beans. The `max-beans-in-free-pool` element controls the size of this pool.

If you are running lots of finders or home methods or creating lots of 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 the Pool Size

Do not change the value of the `max-beans-in-free-pool` parameter 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%, 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.

Tuning Initial Beans in Free Pool

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-bean-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, since initial requests for the bean can be satisfied without generating a new instance.

`initial-bean-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/docs61/ejb/reference.html#initial-beans-in-free-pool_60`.

Setting EJB Caching Size

WebLogic Server allows you to configure the number of active beans that are present in the EJB cache (the in-memory space where beans exist).

The `max-beans-in-cache` element of the `weblogic-ejb-jar.xml` file specifies the maximum number of objects of this class that are allowed in memory. When `max-bean-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.

Using this element sets the cache size for stateful session and entity beans similarly.

For more information, see "Locking and Caching Services for Entity EJBs" at `http://e-docs.bea.com/wls/docs61/ejb/EJB_environment.html#ejb20_locking`.

The `max-beans-in-cache` element is described in *Programming WebLogic Enterprise JavaBeans* at `http://e-docs.bea.com/wls/docs61/ejb/reference.html#max-beans-in-cache_60`.

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 it invokes the `ejbPassivate()` method. When the EJB session object is needed again, it is recalled with the `ejbActivate()` method. When the `ejbPassivate()` call is made, the EJB object is serialized using the Java serialization API or other similar methods and stored in secondary memory (disk). The `ejbActivate()` method causes the opposite.

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 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 for entity EJBs. The WebLogic Server container does this by deferring locking services to the underlying database. Unlike exclusive locking, the underlying data store can provide finer granularity for locking EJB data, in most cases, as well as provide deadlock detection.

For details about database locking, see Locking Services for Entity EJBs, at `http://e-docs.bea.com/wls/docs61/ejb/EJB_environment.fm#LockingServices`.

You specify the locking mechanism used for an EJB by setting the `concurrency-strategy` deployment parameter in `weblogic-ejb-jar.xml`. See `http://e-docs.bea.com/wls/docs61/ejb/reference.fm#concurrency_strategy_60`.

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/docs61/ejb/reference.html#ref_ejbc`.

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

Tuning Parameters for Starting WebLogic Server

Sample scripts are provided with the WebLogic distribution that you can use to start WebLogic Servers, as described in Starting and Stopping WebLogic Servers, at `http://e-docs.bea.com/wls/docs61/adminguide/startstop.html#StartingStoppingServers`.

You will need to modify these scripts to fit your environment and applications. Separate sample scripts are provided for starting the Administration Server and the Managed Server. The scripts for starting the Administration Server are called `startWebLogic.sh` (UNIX) and `startWeblogic.cmd` (Windows). These scripts are located in the configuration subdirectory for your domain.

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\jdk131
  ```

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

  ```
  "%JAVA_HOME%\bin\java" -hotspot –ms512m –mx512m -classpath
  %CLASSPATH% -
  ```

  See “Specifying Heap Size Values” on page 2-7 for details about setting heap size options.

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 WebLogic Server Console

To change your compiler in the console:

1. Start the WebLogic Server Console.
2. Open the Servers folder in the navigation tree.
3. Select your server (`myserver` in a default installation) in the Servers folder.
4. Select the Configuration tab.
5. Select the Compilers tab and enter the full path of the compiler in the Java Compiler text box. For example:

   ```
   c:\visualcafe31\bin\sj.exe
   ```
6. Enter the full path to the JRE rt.jar library in the Append to classpath text box. For example:

   ```
   weblogic_home\jdk131\jre\lib\rt.jar
   ```
7. Click Apply.
8. 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.

The `compileCommand` parameter specifies the Java compiler to use for compiling the generated JSP servlets.

The `precompile` parameter allows you 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` element at `http://e-docs.bea.com/wls/docs61/webapp/weblogic_xml.html#jsp-descriptor`.

Compiling EJB Container Classes

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

For more information, see "WebLogic Server EJB Utilities" at `http://e-docs.bea.com/wls/docs61/ejb/EJB_utilities.html`.

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.

WebLogic Server Clusters and Scalability

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

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, simply add another WebLogic Server 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.

For complete information about clusters, see "Using WebLogic Server Clusters" at `http://e-docs.bea.com/wls/docs61/cluster/index.html`.

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.

Performance Considerations for Multi-CPU Machines

With multi-processor machines, additional consideration must be given to the ratio of clustered WebLogic Server instances to the number of available CPUs. 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, Inc. Sun Enterprise 10000, can potentially host very large clusters or multiple clusters

Before determining the optimal ratio of servers to CPUs, thoroughly test your application to determine:

- *Network Requirements*—If you discover that a web application is primarily network I/O-bound, then you should consider measures to increase network throughput before increasing the number of available CPUs. For truly network I/O-bound applications, installing a faster NIC may increase performance more than additional CPUs, because most CPUs would remain idle while waiting to read available sockets.
- *Disk I/O Requirements*—If you discover that a web application is primarily disk I/O-bound, you should consider upgrading the number of disk spindles or individual disks and controllers before allocating additional CPUs.

In summary, you should ensure that a web application is truly CPU-bound, rather than network or disk I/O-bound, before allocating additional CPUs.

For CPU-bound applications, begin performance tests using a ratio of one WebLogic Server instance for every CPU. If CPU utilization is consistently at or near 100%, increase the ratio of CPUs to servers (for example, allocate one WebLogic Server instance for ever two CPUs). For production systems, keep in mind that some spare CPU cycles should always be available to perform administration tasks.

Although the processing needs of web applications varies, BEA has found that optimal results are generally obtained using a ratio of one Weblogic Server instance for every two CPUs.

Monitoring a WebLogic Server Domain

The tool for monitoring the health and performance of your WebLogic Server domain is the Administration Console. You use the Administration Console to 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 details, see "Monitoring a WebLogic Server Domain" at `http://e-docs.bea.com/wls/docs61/adminguide/monitoring.html`.

4 Tuning WebLogic Server Applications

WebLogic Server only performs as well as the applications running on it. It is important to determine the bottlenecks that impede performance, as described in the following sections:

- "Using Performance Analysis Tools" on page 4-1
- "JDBC Application Tuning" on page 4-2
- "Managing Session Persistence" on page 4-3
- "Minimizing Sessions" on page 4-4
- "Using Execute Queues to Control Thread Usage" on page 4-5

Using Performance Analysis Tools

This section provides 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 API

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

Borland's Optimizeit Performance Profiler is a performance debugging tool for Solaris and Windows.

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

BEA offers three WebLogic jDrivers for use with the WebLogic Server software:

- Type 2 native JDBC driver for Oracle with distributed transaction capability

  See *Installing and Using WebLogic jDriver for Oracle* at `http://e-docs.bea.com/wls/docs61/oracle/index.html`.

- Type 4 JDBC drivers for Informix and Microsoft SQL Server

  See *Installing and Using WebLogic jDriver for Informix* at `http://e-docs.bea.com/wls/docs61/informix4/index.html` and *Installing and Using WebLogic jDriver for Microsoft SQL Server* at `http://e-docs.bea.com/wls/docs61/mssqlserver4/index.html`.

Type-2 drivers use client libraries supplied by a database vendor. Type-4 drivers are pure-Java; they connect to the database server at the wire level without vendor-supplied client libraries.

See "Performance Tuning Your JDBC Application," in *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs61/jdbc/performance.html`.

JDBC Optimization for Type-4 MS SQL Driver

When using the type-4 MS SQL driver, it may be much faster to create and execute an SQL statement either without parameters or with parameter values converted to their string counterparts and added as appropriate to the string, rather than declaring a long series of `setXXX()` calls, followed by `execute()`.

Managing Session Persistence

Optimize your application so that it does as little work as possible when handling session persistence. In WebLogic Server, the following options are available for session persistence:

- "In-Memory Replication" on page 4-3
- "JDBC-based Persistence" on page 4-4

For additional details, see "Configuring Session Persistence," in the *WebLogic Server Administration Guide*, at `http://e-docs.bea.com/wls/docs61/webapp/sessions.html#session-persistence`.

In-Memory Replication

In-memory replication is up to ten times faster than JDBC-based persistence for session state. Use in-memory replication, if possible.

For more information, see "Understanding HTTP Session State Replication," in *Using WebLogic Server Clusters*, at http://`http://e-docs.bea.com/wls/docs61/cluster/servlet.html`.

See also "In-Memory Replication for Stateful Session EJBs," in *Programming WebLogic Enterprise JavaBeans*, at http://`http://e-docs.bea.com/wls/docs61/ejb/EJB_environment.html`.

JDBC-based Persistence

If you are using JDBC-based persistence, optimize your code so that it has as high a granularity for session state persistence as possible. In the case of JDBC-based persistence, every session "put" that you use in your code results in a database write of the entire object.

Keep the number of "puts" that you use during your HTTP session to a mininmum.To minimize how often information is persisted during a given session, look at your "puts" and see if you can combine them into a single, large "put," instead.

For more information, see "Using a Database for Persistent Storage (JDBC Persistence)," in *Assembling and Configuring Web Applications* at `http://e-docs.bea.com/wls/docs61/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.

  Use sessions only for state that cannot realistically be kept on the client or if URL rewriting support is required. Keep simple bits of state, such as a user's name, directly in cookies. You might also write a wrapper class to do the getting and setting of 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.

See "Setting Up Session Management," in *Assembling and Configuring Web Applications*, at `http://e-docs.bea.com/wls/docs61/webapp/sessions.html#session-management`.

Using Execute Queues to Control Thread Usage

You can fine-tune an application's access to execute threads to optimize its performance or limit its CPU utilization by configuring multiple execute queues in WebLogic Server. An execute queue represents a named collection of execute threads that are available to one or more designated servlets, JSPs, EJBs, or RMI objects.

Default WebLogic Server installations are configured with a `default` execute queue, 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.** If you have an application that can potentially consume large amounts of memory, assigning the application 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 deadlocks in 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 for processing the servlet requests, then no threads are available to deliver messages from the JMS queue. In this situation, a deadlock condition exists, and no work can progress. Assigning the servlet to a separate execute queue remedies the potential deadlock, because the servlet and JMS queue do not compete for thread resources.

Execute Queue Drawbacks

Although execute queues can provide fine-tuning for application performance, keep in mind that unused threads represent significant wasted resources in a Weblogic Server system. Without careful consideration of thread usage in all execute queues, you may find that available threads in configured execute queues go unused, while applications in other queues sit idle waiting for threads to become available. In such a situation, the division of threads into queues may yield poorer overall performance than having a single, default execute queue.

Be sure to monitor each execute queue to ensure proper thread usage in the system as a whole. See "Setting Thread Count" on page 3-3 for general information about optimizing the number of threads.

Creating Execute Queues

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 4 execute threads appears in the `config.xml` file as follows:

```
...
<Server
 Name="examplesServer"
 ListenPort="7001"
 NativeIOEnabled="true"/>
 <ExecuteQueue Name="default"
  ThreadCount="15"/>
 <ExecuteQueue Name="CriticalAppQueue"
  ThreadCount="4"/>
 ...
</Server>
```

To configure a new execute queue using the WebLogic Administration Console:

1. Start the Administration Console and click on the name of the server to which you will add the execute queue.
2. Click the Monitoring tab.
3. From the General monitoring tab, click Monitor all Active Queues...
4. Click Configure Execute Queues...
5. Click Configure a New Execute Queue...
6. Enter the name, thread priority, and number of threads for the new queue.
7. Click Create to create the new queue in `config.xml`.

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

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

To assign an EJB object to a configured execute queue, use the `-dispatchPolicy` option with `ejbc`. The `ejbc` compiler passes this option and its argument to `rmic` when compiling the EJB.

A Related Reading

This section provides an extensive performance-related reading list, including the following:

BEA Systems, Inc. Information

- For general information about BEA Systems, see The BEA web site at `http://www.bea.com`.
- BEA WebLogic Server Product Documentation page

  See `http://e-docs.bea.com/wls/docs61/index.html`.
- BEA WebLogic Server White Papers

  See `http://dev2dev.bea.com/products/wlserver61/resources.jsp`.
- *J2EE Design Considerations for WebLogic Server*, BEA White Paper, 2000

  See `http://www.bea.com/products/j2ee_wp_index.shtml`.
- *J2EE Applications and BEA WebLogic Server* by Michael Girdley, Rob Woollen, Sandra Emerson, 2001
- *Professional J2EE Programming with BEA WebLogic Server* by Paco Gomez, Peter Zadrozny, 2000
- J2EE Performance Testing with BEA WebLogic Server by Peter Zadrozny, Philip Aston, and Ted Osborne 2002

Sun Microsystems Information

- For general information about Sun Microsystems, see Sun’s web site at `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`.

- Java 2 SDK, Standard Edition Documentation

  See `http://java.sun.com/products/jdk/1.2/docs`.

- Solaris Tunable Parameters Reference Manual

  See `http://docs.sun.com/db?p=/doc/806-4015`.

- For BEA WebLogic Server and Solaris-specific details, see Sun Microsystems Solaris on SPARC on the BEA Platform page.

  See `http://www.weblogic.com/platforms/sun/index.html`.

- 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, 1997.

- *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

Hewlett-Packard Company Information

- General Hewlett-Packard information

  See `http://www.thenewhp.com`.

- 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 configuration tool

  See `http://www.hp.com/products1/unix/java/java2/hpjconfig/index.html`.

Microsoft Information

- General Microsoft information

  See `http://www.microsoft.com/ms.htm`.
- Windows 2000 Performance Tuning White Paper

  See `http://www.microsoft.com/technet/win2000/win2ksrv/technote/perftune.asp`.
- 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

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

- iPlanet Web Server 4.0 Performance Tuning, Sizing, and Scaling

  See `http://docs.iplanet.com/docs/manuals/enterprise/40/scaling/perf.htm`.

- 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.

Network Performance Tools

- Systems, Software, Technology (SST) Incorporated, Trace Plus/Ethernet.

  TracePlus/Ethernet is a network packet analysis tool for Windows 95/98/ME, NT 4.x, Windows 2000/XP.

See `http://www.sstinc.com/home.html`.

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 Performance Profiler from Borland, a performance debugging tool for Solaris and Windows.

  See `http://www.borland.com/optimizeit/optimizeit_profiler/index.com`.

- 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`.

- 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/`.

- SE Toolkit, a performance analysis tool kit.

  See `http://www.setoolkit.com/`.

Benchmarking Information

- SPECjbb2000

SPECjbb2000 is a software benchmark product developed by the Standard Performance Evaluation Corporation (SPEC). It 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 Specification

  See `http://jcp.org/aboutJava/communityprocess/final/jsr004/index.html`

- eTesting Labs Inc.

  eTesting Labs Inc., a Ziff Davis Media company, is an independent developer of benchmark software, including WebBench 4.0.

  See `http://www.etestinglabs.com/`.

Java Virtual Machine (JVM) Information

- JVM Corner at artima.com

  See `http://www.artima.com/jvm`.

- Frequently asked questions about the Java HotSpot virtual machine

  This Sun Microsystems FAQ answers common questions about Java HotSpot technology and about performance in general.

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

- Tuning Garbage Collection with the 1.3.1 Java Virtual Machine

  This Sun Microsystems document provides an thorough overview of garbage collection tuning.

  See `http://java.sun.com/docs/hotspot/gc/`.

- Java HotSpot VM Options

  This Sun Microsystems 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`.

- The Java HotSpot Client and Server Virtual Machines

  This Sun Microsystems document discusses the two implementations of the Java virtual machine that are available for J2SE 1.3

  See `http://java.sun.com/j2se/1.3/docs/guide/performance/hotspot.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/docs61/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

  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/docs61/jms/index.html`.

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

  See `http://dev2dev.bea.com/resourcelibrary/whitepapers/index.jsp#Server`

- JMS Specification

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

General Performance Information

- Jack Shirazi's Java Performance Tuning web site.

  See `http://www.javaperformancetuning.com`.

- The Software Testing and Quality Engineering Magazine, Web Application Scalability, "Avoiding Scalability Shock" by Bill Shea, May/June 2000.

  See `http://www.stqemagazine.com/index.asp?frame=CORE&content=BACKISSUE&stamp=417165320).`

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

Index