BEA WebLogic Server

Performance and Tuning

BEA WebLogic Server 6.0
Document Date: August 22, 2001

BEA WebLogic Server Performance and Tuning

Part Number	Document Date	Software Version
N/A	August 22, 2001	BEA WebLogic Server Version 6.0

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 capacity and configuration, 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 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 1-2
- "Operating System Tuning" on page 1-3
- "Network Performance" on page 1-4

Hardware Tuning

When you examine performance, consider your hardware capacity and configuration first. BEA certifies WebLogic Server on multiple hardware platforms. BEA only certifies platforms that pass rigorous internal testing. Table 1-1 presents links to further information about hardware tuning.

Table 1-1 Hardware Considerations

Issue	For more information
Platform Support Page	The Platform Support page at `http://edocs.bea.com/wls/platforms/index.html` is frequently updated and contains the latest certification information on various platforms. Check this page often for the most current platform information.
Solaris	For BEA WebLogic Server and Solaris-specific details, see "Sun Microsystems Solaris on SPARC" on the Platform Support page at `http://www.weblogic.com/platforms/sun/index.html`. See also "Sun Microsystems Information" on page A-2.
Hewlett Packard	For BEA WebLogic Server and HP-UX-specific details, see "WebLogic HP-UX Platform Support" on the Platform Support page at `http://www.weblogic.com/platforms/hpux/index.html.` See also "Hewlett-Packard Company Information" on page A-3.
Standardized benchmarks and metrics	The Standard Performance Evaluation Corporation provides a set of standardized benchmarks and metrics for evaluating computer system performance. See www.spec.org.

Operating System Tuning

Tune your operating system according to your operating system documentation. BEA certifies WebLogic Server on multiple operating systems. Table 1-2 presents links to further information about operating system tuning.

Table 1-2 Operating System Considerations

Issue	For more information
Platform Support Page	The Platform Support page at `http://e-docs.bea.com/wls/docs60/platforms/index.html` is frequently updated and contains the latest certification information on various platforms. Check this page often for the most current operating system information.
File descriptors	On the UNIX platform, each socket connection to the server consumes a file descriptor. You need to configure your operating system to have the appropriate number of file descriptors. See "Tuning Solaris File Descriptor Limits" on the "Sun Microsystems Solaris on SPARC" Platform Support page at `http://e-docs.bea.com/wls/docs60/platforms/sun/index.html`.
Solaris TCP tuning parameters	See "Setting Solaris Tunable Parameters" on the "Sun Microsystems Solaris on SPARC" Platform Support page at `http://e-docs.bea.com/wls/docs60/platforms/sun/index.html`.
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. For more information, see your operating system documentation.

Table 1-2 Operating System Considerations (Continued)

Issue	For more information
Using native I/O for serving static files (Windows only)	When running WebLogic Server on Windows NT/2000 you can specify that WebLogic Server use the native operating system call `TransmitFile` instead of using Java methods to serve static files such as HTML files, text files, and image files. Using native I/O can provide performance improvements when serving larger static files. See "Using Native I/O for Serving Static Files" in the *BEA WebLogic Server Administration Guide* for more information. See `http://http://e-docs.bea.com/wls/docs60/adminguide/web_server.html#nativeio`.

Network Performance

Network performance is affected when the supply of resources is unable to keep up with the demand for resources. Table 1-3 presents some issues to consider.

Table 1-3 Network Configuration Considerations

Issue	Consideration
Network hardware and software	If you have a problem with one or more network components (hardware or software), you must isolate and eliminate the problem. For more information, see your network administrator.
Network bandwidth	You must 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. For more information, see your network administrator.

Table 1-3 Network Configuration Considerations (Continued)

Issue	Consideration
LAN infrastructure	If you have high network traffic that consistently exceeds the capacity of the available resources (for example, the hit rate on a Web server has reached its maximum value while the system is 100 percent busy), you must either: ■ redesign the network and redistribute the load, ■ reduce the number of network clients, ■ or increase the number of systems handling the network load. For more information, see your network administrator.

CHAPTER

2 Tuning Java Virtual Machines (JVMs)

The Java virtual machine (JVM) is is an abstract computing machine designed to support the Java programming language. How you tune your JVM affects the performance of WebLogic Server and your applications.The following sections discuss JVM tuning considerations:

- "JVM Tuning Considerations" on page 2-2
- "Tuning Heap Size" on page 2-3
- "Using Non-Standard Java Command Line Options" on page 2-6

JVM Tuning Considerations

Table 2-1 discusses general JVM tuning considerations.

Table 2-1 General JVM Tuning Considerations

Issue	Description
JVM vendor and version	The JVM vendor and version affect performance. 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://edocs.bea.com/wls/platforms/index.html` is frequently updated and contains the latest certification information on various platforms. Check this page often for the most current JVM information.
Mixed client/server JVMs	Deployments using different JVM versions for the client and server are supported in WebLogic 6.0. For information about support for mixed client/server JVMs, see `http://e-docs.bea.com/wls/platforms/index.html#mix`. Check the Platform Support page often for the most current information.
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 because the Java HotSpot Client and Server Virtual Machines are two separate JITs accessing the same runtime system. See "Java Virtual Machine (JVM) Information" on page A-6.*

Table 2-1 General JVM Tuning Considerations (Continued)

Issue	Description
UNIX threading models	On UNIX, two threading models are available: green threads and native threads. To get the best performance and scalability with WebLogic Server, choose a JVM that uses native threads, not green threads. See your VM documentation for more information about threading. 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`.
Tuning heap size	See "Tuning Heap Size" on page 2-3.

Tuning Heap Size

The JVM heap size determines how often and how long the VM spends collecting garbage (de-allocating unused Java objects).The JVM heap is a repository for live objects, dead objects, and free memory. When a JVM runs out of memory in the heap, all execution in the JVM stops while a garbage collection algorithm goes through memory and frees space that is no longer required by an application. This process affects performance because server-side work cannot proceed during garbage collection.

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.

Determining Heap Size

This section describes how to determine the most effective heap size:

1. Monitor the performance of WebLogic Server under maximum load while running your application.
2. Redirect `-verbosegc` messages to a file.
   - On HPUX, the following non-standard option redirects `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 it
   - On Solaris, use the following command:

     ```
     weblogic.Server > server.out 2>&1
     ```
3. Use the `-verbosegc` option to measure exactly how much time and resources are put into garbage collection.

   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.
4. 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.

5. 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.

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

Setting Minimum Heap Size Equal to Maximum Heap Size

Set the minimum heap size equal to the maximum heap size. This yields higher performance throughput than setting the values differently, and prevents the JVM from spending time incrementing the heap. You specify the minimum and maximum values for Java heap memory when starting the WebLogic Administration Server from the Java command line.

For example:

```
$ java ... -XX:NewSize=128m -XX:MaxNewSize=128m
-XX:SurvivorRation=2 -Xms512m -Xmx512m
```

Setting High Heap Size for Benchmarking

To ensure maximum performance, you might set high heap size values to ensure that garbage collection does not occur during the entire run of the benchmark.

Using Non-Standard Java Command Line Options

You can use non-standard `java` options to improve performance. Be aware that use of these options depends on how your application is coded.

Non-Standard Options for Hotspot VM on NT

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

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

Option	Description
`-Xnoclassgc`	This option disables garbage collection for the class, preventing re-loading of the class when it is referenced after all references to it have been lost. This option requires a greater heap size.
`-XX:NewSize`, `-XX:MaxNewSize`, `-XX:NewSizeThreadIncrease`	These options control the amount of heap reserved for object creation. ■ `-XX:NewSize` sets the initial size. ■ `-XX:MaxNewSize` sets the maximum size. ■ `-XX:NewSizeThreadIncrease` sets the increment-size. For applications that frequently create a lot of objects modifying the values of these options could boost performance.
`-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 2-2 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`.

Non-Standard Options for HotSpot VM on 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.

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` 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\jdk130\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.

For more information about setting your server's java compiler in the `weblogic.xml` file, see the description of the `jsp-descriptor` element, in *Developing WebLogic Server Applications* at `http://e-docs.bea.com/wls/docs60/programming/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/docs60/ejb/EJB_utilities.html`.

Precompiling JSPs

You can configure WebLogic Server to precompile your JSPs when WebLogic Server starts up by defining the `weblogic.jsp.precompile` context parameter in the `web.xml` deployment descriptor.

For more information on the `web.xml` deployment descriptor, see "Writing Web Application Deployment Descriptors", at `http://e-docs.bea.com/wls/docs60/programming/webappdeployment.html`.

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/docs60/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.

Setting Thread Pool Size

The value of the `ThreadPoolSize` attribute equals the number of simultaneous operations that can be performed by WebLogic Server. You can degrade performance by increasing this value unnecessarily.

`ThreadPoolSize` is an attribute of the `Server` element in the `config.xml` file.The value of the `ThreadPoolSize` attribute equals the number of simultaneous operations that can be performed by WebLogic Server. As work enters a WebLogic Server, it is placed in an execute queue. This work is then assigned to a thread that does the work on it.

The default number of threads is 15. Threads consume resources, so handle this attribute with care. For most applications, leave the default value unchanged.

Modifying the Thread Pool Size

Adding more threads 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 this value unnecessarily. Because threads are resources that consume memory, a very high execute thread count causes more memory to be used and increases context switching. This degrades your performance.

The value of the `ThreadPoolSize` attribute depends very much on the type of work the application does. For example, if your client application is thin and does a lot of its work through remote invocation, the time your client application spends connected will be greater than for a client application that does a lot of client-side processing, for example.

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 Pool Size Scenarios

Table 3-1 shows some possible scenarios.

Table 3-1 Thread Pool Size Scenarios

When...	Results	Do This:
Thread Pool Size < number of CPUs	Results in an under-utilized CPU.	Increase the thread count.

Table 3-1 Thread Pool Size Scenarios (Continued)

When...	Results	Do This:
(Thread Pool Size == number of CPUs)	Theoretically ideal, but the CPUs are under-utilized.	Increase the thread count.
(Thread Pool Size > 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.
(Thread Pool Size > 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 Pool Size Too Low

If your Thread Pool Size is too low, these symptoms appear when WebLogic Server is running under maximum load:

- CPU is waiting to do work, but there is work that could be done.
- Cannot get 100% CPU.

Symptoms: Thread Pool Size Too High

If your Thread Pool Size is set too high when running WebLogic Server under maximum load, your performance increases as you decrease the number of threads.

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. The optimal value for this property is application-specific. The default is value 33, and the valid range is 1-99.

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 3-10.

The `ThreadPoolPercentSocketReaders` attribute in the `config.xml` file 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. The optimal value for this attribute is very application-specific. 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.

Connection Backlog Buffering

The `AcceptBacklog` attribute in the `config.xml file` allows you to set the number of connections available for the TCP backlog.

In the Administration Console, select Server→Configuration→Tuning, and 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 TCP backlog attribute (`AcceptBacklog`) could be set too low. 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.

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

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 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.

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.

Do not change this value 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, then increasing this parameter does not significantly improve performance. For applications where EJBs are very 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.

For more information about EJB Pools, see “Programming WebLogic Enterprise JavaBeans” at `http://e-docs.bea.com/wls/docs60/ejb/EJB_environment.html`.

The `max-beans-in-free-pool` element is described in “WebLogic Server 6.0 EJB Deployment Properties” at `http://e-docs.bea.com/wls/docs60/ejb/reference.html`.

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 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. For more information, see "Locking and Caching Services for Entity EJBs" at `http://e-docs.bea.com/wls/docs60/ejb/EJB_environment.html#ejb20_locking`.

Activation and Passivation of Stateful Session EJBs

Activation and passivation of stateful session EJBs is analogous to virtual memory on a computer. You can minimize the number of times that your beans are activated and passivated, by setting the appropriate cache size using the `max-beans-in-cache` attribute.

When a bean is brought into the cache, `ejbActivate()` is called, when it is removed, `ejbPassivate()` is called. It is basically equivalent to virtual memory being kept in memory or on disk. Tuning `max-beans-in-cache` too high consumes memory unnecessarily.

The goal for setting an optimal value for `max-beans-in-cache` is to avoid excessive passivation (the transfer of an EJB instance from memory to secondary storage) and activation (the transfer of an EJB instance from secondary storage to memory). The EJB container performs passivation when it invokes the `ejbPassivate()` method and 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 just 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.

Determining Cache Size

For more information about the `max-beans-in-cache` element, see "WebLogic Server 6.0 EJB Deployment Properties" at `http://e-docs.bea.com/wls/docs60/ejb/reference.html`.

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/docs60/adminguide/monitoring.html`.

Using WebLogic Server Performance Packs

Benchmarks show performance improvements in WebLogic Server of up to a factor of three for most workloads, 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://edocs.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.

CHAPTER

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 a Profiler to Reveal Hot Spots" on page 4-1
- "JDBC Application Tuning" on page 4-2
- "Setting Transaction Isolation Level" on page 4-2
- "Managing Session Persistence" on page 4-3
- "Minimizing Sessions" on page 4-4

Using a Profiler to Reveal Hot Spots

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. See "Performance Analysis Tools" on page A-4 for a list of common profilers.

JDBC Application Tuning

See "Performance Tuning Your JDBC Application," in *Programming WebLogic JDBC* at `http://e-docs.bea.com/wls/docs60/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 without parameters, or with parameter values converted to their string counterparts and added as appropriate to the string, rather than to do a long series of `setXXX()` calls, followed by `execute()`.

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/docs60/ejb/reference.html#ref_ejbc`.

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

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 4-3

For additional details, see "Configuring Session Persistence," in the *WebLogic Server Administration Guide*, at `http://e-docs.bea.com/wls/docs60/adminguide/config_web_app.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://e-docs.bea.com/wls/docs60cluster/servlet.html`.

See also "In-Memory Replication for Stateful Session EJBs," in *Programming WebLogic Enterprise JavaBeans*, at `http://e-docs.bea.com/wls/docs60/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. This means that you want to reduce the number of "puts" that you use during your HTTP session. In the case of JDBC-based persistence, every session "put" that you do results in a database write. You want to minimize how often information is persisted during a given session. Look at your "puts" and see if you can combine them into one large put instead.

For more information, see "Using a Database for Persistent Storage," in the *WebLogic Server Administration Guide* at `http://e-docs.bea.com/wls/docs60/adminguide/config_web_app.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.

For more information, see "Setting Up Session Management," in the *WebLogic Server Administration Guide*, at `http://e-docs.bea.com/wls/docs60/adminguide/config_web_app.html#session-management`.

A Related Reading

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

- "BEA Systems, Inc. Information" on page A-1
- "Sun Microsystems Information" on page A-2
- "Hewlett-Packard Company Information" on page A-3
- "Microsoft Information" on page A-3
- "Network Performance Tools" on page A-4
- "Performance Analysis Tools" on page A-4
- "Benchmarking Information" on page A-5
- "Java Virtual Machine (JVM) Information" on page A-6
- "Enterprise JavaBeans Information" on page A-7
- "General Performance Information" on page A-7

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/docs60`.

- BEA WebLogic Server White Papers

  See `http://www.bea.com/products/weblogic/server/papers.shtml`.
- *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.

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/ab2/coll.709.2/SOLTUNEPARAMREF/`.
- 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.

Hewlett-Packard Company Information

- General Hewlett-Packard information

  See `http://www.hp.com`.
- Java products for HP-UX

  See `http://www.hp.com/products1/unix/java/index.html`.
- Java Performance Tuning on HP-UX

  See `http://www.devresource.hp.com/JavaATC/JavaPerfTune/`.
- glance/jpm performance diagnostic tool

  See `http://devresource.hp.com/JavaATC/JavaPerfTune/tools.html#glance`.
- 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`.

Network Performance Tools

- Systems, Software, Technology (SST) Incorporated, TracePlus/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 Java Performance Profiler

  A performance debugging tool for Solaris and NT.

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

- Hewlett Packard JMeter

  A Hewlett Packard tool for analyzing profiling information.

See `http://www.hp.com/products1/unix/java/hpjmeter/`.

- 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`.

- Topaz, Mercury Interactive's application performance management solution

  See `http://www-svca.mercuryinteractive.com/products/topaz/`.

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

  ECperf is an EJB performance workload that is real-world, scalable and captures the essence of why component models exist.

  See `http://jsp2.java.sun.com/aboutJava/communityprocess/jsr/jsr_004_ecperf.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.zdnet.com/etestinglabs/filters/benchmarks/`.

- SE Toolkit

  A performance analysis tool kit.

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

Java Virtual Machine (JVM) Information

- 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`.

- JavaWorld article, Sun Microsystems HotSpot

  This document from JavaWorld discusses Sun’s HotSpot VM.

  See `http://www.javaworld.com/jw-03-1998/jw-03-hotspot.html`.

- Which Java VM scales best?

  From JavaWorld, results of a new VolanoMark 2.0 server benchmark show how 12 virtual machines stack up.

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

Enterprise JavaBeans Information

- *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://www2.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/`.

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)`.
- *High-Performance Java Platform Computing*™ by Thomas W. Christopher, George K. Thiruvathukal, 2000.

  See `http://www.toolsofcomputing.com/JavaThreads/`.
- *Performance and Idiom Guide* by Craig Larman and Rhett Guthrie, 1999.

Index