**Tuning WebLogic Server**

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

* [Setting Your Java Compiler](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104255)

* [WebLogic Server Clusters and Scalability](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104298)

* [Setting Thread Pool Size](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104317)

* [Allocating Threads to Act as Socket Readers](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104408)

* [Connection Backlog Buffering](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104418)

* [Setting EJB Pool Size](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104431)

* [Setting EJB Caching Size](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104446)

* [Monitoring a WebLogic Server Domain](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104485)

* [Using WebLogic Server Performance Packs](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104504)

**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

1. 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

1. Click Apply.
2. 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](http://docs.oracle.com/cd/E13222_01/wls/docs60/programming/weblogic_xml.html#jsp-descriptor) element, in *Developing WebLogic Server Applications*.

**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"](http://docs.oracle.com/cd/E13222_01/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"](http://docs.oracle.com/cd/E13222_01/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"](http://docs.oracle.com/cd/E13222_01/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](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html" \l "1104349) 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. |
| (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](http://docs.oracle.com/cd/E13222_01/wls/docs60/perform/WLSTuning.html#1104504).

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 Serverhttp://docs.oracle.com/cd/E13222_01/wls/docs60/images/chars/arrwrite.gifConfigurationhttp://docs.oracle.com/cd/E13222_01/wls/docs60/images/chars/arrwrite.gifTuning, 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. TheAcceptBacklog 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"](http://docs.oracle.com/cd/E13222_01/wls/docs60/ejb/EJB_environment.html).

The [max-beans-in-free-pool](http://docs.oracle.com/cd/E13222_01/wls/docs60/ejb/reference.html) element is described in "WebLogic Server 6.0 EJB Deployment Properties".

**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"](http://docs.oracle.com/cd/E13222_01/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. Tuningmax-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](http://docs.oracle.com/cd/E13222_01/wls/docs60/ejb/reference.html) element, see "WebLogic Server 6.0 EJB Deployment Properties".

**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"](http://docs.oracle.com/cd/E13222_01/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.
2. Choose Edithttp://docs.oracle.com/cd/E13222_01/wls/docs60/images/chars/arrwrite.gifFind 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.