

GlassFish Server Performance Tuning Guide

3 Tuning the GlassFish Server

This chapter describes some ways to tune your GlassFish Server installation for optimum performance.

Note that while this chapter describes numerous interactions with both the GlassFish Server Administration Console and the command-line interface, it is not intended to provide exhaustive descriptions of either. For complete information about using the Administration Console, refer to the Administration Console online help. For complete information about using the GlassFish Server command-line interface, refer to the other titles in the GlassFish Server documentation set at

https://download.oracle.com/docs/cd/E18930_01/index.htm 

(https://download.oracle.com/docs/cd/E18930_01/index.htm) .

The following topics are addressed here:

- Using the GlassFish Server Performance Tuner (#gkxwm)
- Deployment Settings (#abedo)
- Logger Settings (#abeds)
- Web Container Settings (#abedw)
- EJB Container Settings (#abeea)
- Java Message Service Settings (#abeel)
- Transaction Service Settings (#abeem)
- HTTP Service Settings (#abeet)
- Network Listener Settings (#abegk)
- Transport Settings (#gkxjt)
- Thread Pool Settings (#abehk)



- ORB Settings (#abegt)
- Resource Settings (#abehp)
- Load Balancer Settings (#gkxvd)

Using the GlassFish Server Performance Tuner

You can significantly improve the performance of GlassFish Server and the applications deployed on it by adjusting a few deployment and server configuration settings. These changes can be made manually, as described in this chapter, or by using the built-in Performance Tuner in the GlassFish Server Administration Console.

The Performance Tuner recommends server settings to suit the needs of your GlassFish Server deployment. It helps you reach an optimal configuration, although finer tuning might be needed in case of specific requirements. You can configure performance tuning for the entire domain, or for individual GlassFish Server instances or clusters. The Tuner performs a static analysis of GlassFish Server resources and throughput requirements. Note that no dynamic inspection of the system is performed.

For complete information about using the Performance Tuning features available through the GlassFish Server Administration Console, refer to the Administration Console online help. You may also want to refer to the following resources for additional information:

- To view a demonstration video showing how to use the GlassFish Server Performance Tuner, see the Oracle GlassFish Server 3.1 - Performance Tuner demo  (<http://www.youtube.com/watch?v=FavsE2pzAjc>) .
- To find additional Performance Tuning development information, see the Performance Tuner in Oracle GlassFish Server 3.1  (http://blogs.oracle.com/jenblog/entry/performance_tuner_in_oracle_glassfish) blog.

Deployment Settings

Deployment settings can have significant impact on performance. Follow these guidelines when configuring deployment settings for best performance:

- Disable Auto-Deployment (#abedp)
- Use Pre-compiled JavaServer Pages (#abedq)
- Disable Dynamic Application Reloading (#abedr)

Disable Auto-Deployment

Enabling auto-deployment will adversely affect deployment, though it is a convenience in a development environment. For a production system, disable auto-deploy to optimize performance. If auto-deployment is enabled, then the Reload Poll Interval setting can have a significant performance impact.

To enable or disable auto-deployment from the GlassFish Server Administration Console, navigate to the Domain node and then click the Applications Configuration tab. Refer to the Administration Console for further instructions. Alternatively, refer to "To Deploy an Application or Module Automatically ([../doc.312/e24929/deploying-applications.htm#GSDPG00041](http://doc.312/e24929/deploying-applications.htm#GSDPG00041))" in *Oracle GlassFish Server Application Deployment Guide* for instructions on enabling or disabling auto-deployment.

Use Pre-compiled JavaServer Pages

Compiling JSP files is resource intensive and time consuming. Pre-compiling JSP files before deploying applications on the server will improve application performance. When you do so, only the resulting servlet class files will be deployed.

You can specify to precompile JSP files when you deploy an application through the Administration Console or `deploy` subcommand. You can also specify to pre-compile JSP files for a deployed application with the Administration Console. Navigate to the Domain node and then click the Applications Configuration tab. Refer to the Administration Console for further instructions.

Disable Dynamic Application Reloading

If dynamic reloading is enabled, the server periodically checks for changes in deployed applications and automatically reloads the application with the changes. Dynamic reloading is intended for development environments and is also incompatible with session persistence. To improve performance, disable dynamic class reloading.

You can use the Administration Console to disable dynamic class reloading for an application that is already deployed. Navigate to the Domain node and then click the Applications Configuration tab. Refer to the Administration Console for further instructions.

Logger Settings

The GlassFish Server produces writes log messages and exception stack trace output to the log file in the logs directory of the instance, *domain-dir* / `logs` . The volume of log activity can impact server performance; particularly in benchmarking situations.

General Settings

In general, writing to the system log slows down performance slightly; and increased disk access (increasing the log level, decreasing the file rotation limit or time limit) also slows down the application.

Also, make sure that any custom log handler does not log to a slow device like a network file system since this can adversely affect performance.

Log Levels

Set the log level for the server and its subsystems in the GlassFish Server Administration Console. Navigate to the Configurations>*configuration-name*>Logger Settings page, and follow the instructions in the online help. Alternatively, you can configure logging by following the instructions in "Administering the Logging Service ([../doc.312/e24928/logging.htm#GSADG00010](#)) " in *Oracle GlassFish Server Administration Guide*.

Web Container Settings

Set Web container settings in the GlassFish Server Administration Console by navigating to the Configurations>*configuration-name*>Web Container node. Follow the instructions in the online help for more information. Alternatively, you can configure Web container settings by following the instructions in "Administering Web Applications ([../doc.312/e24928/webapps.htm#GSADG00009](#)) " in *Oracle GlassFish Server Administration Guide*.

- Session Properties: Session Timeout (#abedx)
- Manager Properties: Reap Interval (#abedy)
- Disable Dynamic JSP Reloading (#abedz)

Session Properties: Session Timeout

Session timeout determines how long the server maintains a session if a user does not explicitly invalidate the session. The default value is 30 minutes. Tune this value according to your application requirements. Setting a very large value for session timeout can degrade performance by causing the server to maintain too many sessions in the session store. However, setting a very small value can cause the server to reclaim sessions too soon.

Manager Properties: Reap Interval

Modifying the reap interval can improve performance, but setting it without considering the nature of your sessions and business logic can cause data inconsistency, especially for time-based persistence-frequency.

For example, if you set the reap interval to 60 seconds, the value of session data will be recorded every 60 seconds. But if a client accesses a servlet to update a value at 20 second increments, then inconsistencies will result.

For example, consider the following online auction scenario:

- Bidding starts at \$5, in 60 seconds the value recorded will be \$8 (three 20 second intervals).
- During the next 40 seconds, the client starts incrementing the price. The value the client sees is \$10.
- During the client's 20 second rest, the GlassFish Server stops and starts in 10 seconds. As a result, the latest value recorded at the 60 second interval (\$8) is be loaded into the session.

- The client clicks again expecting to see \$11; but instead sees is \$9, which is incorrect.
- So, to avoid data inconsistencies, take into the account the expected behavior of the application when adjusting the reap interval.

Disable Dynamic JSP Reloading

On a production system, improve web container performance by disabling dynamic JSP reloading. To do so, edit the `default-web.xml` file in the `config` directory for each instance. Change the servlet definition for a JSP file to look like this:

```
<servlet> <servlet-name>jsp</servlet-name> <servlet-
class>org.apache.jasper.servlet.JspServlet</servlet-class> <init-param> <param-
name>development</param-name> <param-value>>false</param-value> </init-param>
<init-param> <param-name>xpoweredBy</param-name> <param-value>>true</param-value>
</init-param> <init-param> <param-name>genStrAsCharArray</param-name> <param-
value>>true</param-value> </init-param> <load-on-startup>3</load-on-startup>
</servlet>
```

EJB Container Settings

The EJB Container has many settings that affect performance. As with other areas, use monitor the EJB Container to track its execution and performance.

You can configure most EJB container settings from the GlassFish Server Administration Console by navigating to the Configurations>*configuration-name*>EJB Container node and then following the instructions in the online help.

Monitoring the EJB Container

Monitoring the EJB container is disabled by default. You can enable EJB monitoring through the GlassFish Server Administration Console by navigating to the the Configurations>*configuration-name*>Monitoring node and then following the instructions in the online help. Set the monitoring level to LOW for to monitor all deployed EJB components, EJB pools, and EJB caches. Set the monitoring level to HIGH to also monitor EJB business methods.

Tuning the EJB Container

The EJB container caches and pools EJB components for better performance. Tuning the cache and pool properties can provide significant performance benefits to the EJB container.

The pool settings are valid for stateless session and entity beans while the cache settings are valid for stateful session and entity beans.

The following topics are addressed here:

- Overview of EJB Pooling and Caching (#abeed)
- Tuning the EJB Pool (#abeee)
- Tuning the EJB Cache (#abeeg)
- Pool and Cache Settings for Individual EJB Components (#abeei)
- Commit Option (#abeej)

Overview of EJB Pooling and Caching

Both stateless session beans and entity beans can be pooled to improve server performance. In addition, both stateful session beans and entity beans can be cached to improve performance.

Table 3-1 Bean Type Pooling or Caching

Bean Type	Pooled	Cached
Stateless Session	Yes	No
Stateful Session	No	Yes
Entity	Yes	Yes

The difference between a pooled bean and a cached bean is that pooled beans are all equivalent and indistinguishable from one another. Cached beans, on the contrary, contain conversational state in the case of stateful session beans, and are associated with a primary key in the case of entity beans. Entity beans are removed from the pool and added to the cache on `ejbActivate()` and removed from the cache and added to the pool on `ejbPassivate()`. `ejbActivate()` is called by the container when a needed entity bean is not in the cache. `ejbPassivate()` is called by the container when the cache grows beyond its configured limits.

Note:

If you develop and deploy your EJB components using Oracle Java Studio, then you need to edit the individual bean descriptor settings for bean pool and bean cache. These settings might not be suitable for production-level deployment.

Tuning the EJB Pool

A bean in the pool represents the pooled state in the EJB lifecycle. This means that the bean does not have an identity. The advantage of having beans in the pool is that the time to create a bean can be saved for a request. The container has mechanisms that create pool objects in the background, to save the time of bean creation on the request path.

Stateless session beans and entity beans use the EJB pool. Keeping in mind how you use stateless session beans and the amount of traffic your server handles, tune the pool size to prevent excessive creation and deletion of beans.

EJB Pool Settings

An individual EJB component can specify cache settings that override those of the EJB container in the `<bean-pool>` element of the EJB component's `sun-ejb-jar.xml` deployment descriptor.

The EJB pool settings are:

- **Initial and Minimum Pool Size:** the initial and minimum number of beans maintained in the pool. Valid values are from 0 to `MAX_INTEGER`, and the default value is 8. The corresponding EJB deployment descriptor attribute is `steady-pool-size`.

Set this property to a number greater than zero for a moderately loaded system. Having a value greater than zero ensures that there is always a pooled instance to process an incoming request.

- **Maximum Pool Size:** the maximum number of connections that can be created to satisfy client requests. Valid values are from zero to `MAX_INTEGER`, and the default is 32. A value of zero means that the size of the pool is unbounded. The potential implication is that the JVM heap will be filled with objects in the pool. The corresponding EJB deployment descriptor attribute is `max-pool-size`.

Set this property to be representative of the anticipated high load of the system. An very large pool wastes memory and can slow down the system. A very small pool is also inefficient due to contention.

- **Pool Resize Quantity:** the number of beans to be created or deleted when the cache is being serviced by the server. Valid values are from zero to `MAX_INTEGER` and default is 16. The corresponding EJB deployment descriptor attribute is `resize-quantity`.

Be sure to re-calibrate the pool resize quantity when you change the maximum pool size, to maintain an equilibrium. Generally, a larger maximum pool size should have a larger pool resize quantity.

- **Pool Idle Timeout:** the maximum time that a stateless session bean, entity bean, or message-driven bean is allowed to be idle in the pool. After this time, the bean is destroyed if the bean in case is a stateless session bean or a message driver bean. This is a hint to server. The default value is 600 seconds. The corresponding EJB deployment descriptor attribute is `pool-idle-timeout-in-seconds`.

If there are more beans in the pool than the maximum pool size, the pool drains back to initial and minimum pool size, in steps of pool resize quantity at an interval specified by the pool idle timeout. If the resize quantity is too small and the idle timeout large, you will not see the pool draining back to steady size quickly enough.

Tuning the EJB Cache

A bean in the cache represents the ready state in the EJB lifecycle. This means that the bean has an identity (for example, a primary key or session ID) associated with it.

Beans moving out of the cache have to be passivated or destroyed according to the EJB lifecycle. Once passivated, a bean has to be activated to come back into the cache. Entity beans are generally stored in databases and use some form of query language semantics to load and store data. Session beans have to be serialized when storing them upon passivation onto the disk or a database; and similarly have to be deserialized upon activation.

Any incoming request using these "ready" beans from the cache avoids the overhead of creation, setting identity, and potentially activation. So, theoretically, it is good to cache as many beans as possible. However, there are drawbacks to caching:

- Memory consumed by all the beans affects the heap available in the Virtual Machine.
- Increasing objects and memory taken by cache means longer, and possibly more frequent, garbage collection.
- The application server might run out of memory unless the heap is carefully tuned for peak loads.

Keeping in mind how your application uses stateful session beans and entity beans, and the amount of traffic your server handles, tune the EJB cache size and timeout settings to minimize the number of activations and passivations.

EJB Cache Settings

An individual EJB component can specify cache settings that override those of the EJB container in the `<bean-cache>` element of the EJB component's `sun-ejb-jar.xml` deployment descriptor.

The EJB cache settings are:

- **Max Cache Size:** Maximum number of beans in the cache. Make this setting greater than one. The default value is 512. A value of zero indicates the cache is unbounded, which means the size of the cache is governed by Cache Idle Timeout and Cache Resize Quantity. The corresponding EJB deployment descriptor attribute is `max-cache-size`.
- **Cache Resize Quantity:** Number of beans to be created or deleted when the cache is serviced by the server. Valid values are from zero to `MAX_INTEGER`, and the default is 16. The corresponding EJB deployment descriptor attribute is `resize-quantity`.

- **Removal Timeout:** Amount of time that a stateful session bean remains passivated (idle in the backup store). If a bean was not accessed after this interval of time, then it is removed from the backup store and will not be accessible to the client. The default value is 60 minutes. The corresponding EJB deployment descriptor attribute is `removal-timeout-in-seconds`.
- **Removal Selection Policy:** Algorithm used to remove objects from the cache. The corresponding EJB deployment descriptor attribute is `victim-selection-policy`. Choices are:
 - NRU (not recently used). This is the default, and is actually pseudo-random selection policy.
 - FIFO (first in, first out)
 - LRU (least recently used)
- **Cache Idle Timeout:** Maximum time that a stateful session bean or entity bean is allowed to be idle in the cache. After this time, the bean is passivated to the backup store. The default value is 600 seconds. The corresponding EJB deployment descriptor attribute is `cache-idle-timeout-in-seconds`.
- **Refresh period:** Rate at which a read-only-bean is refreshed from the data source. Zero (0) means that the bean is never refreshed. The default is 600 seconds. The corresponding EJB deployment descriptor attribute is `refresh-period-in-seconds`. Note: this setting does not have a custom field in the Admin Console. To set it, use the Add Property button in the Additional Properties section.

Pool and Cache Settings for Individual EJB Components

Individual EJB pool and cache settings in the `sun-ejb-jar.xml` deployment descriptor override those of the EJB container. The following table lists the cache and pool settings for each type of EJB component.

Cache or Pool Setting	Stateful Session Bean	Stateless Session Bean	Entity Bean	Entity Read-Only Bean	Message Driven Bean
<code>cache-resize-quantity</code>	X		X	X	
<code>max-cache-size</code>	X		X	X	
<code>cache-</code>	X		X	X	

idle- timeout-in- seconds					
removal- timeout-in- seconds	X		X	X	
victim- selection- policy	X		X	X	
refresh- period-in- seconds				X	
steady- pool-size		X	X	X	
pool- resize- quantity		X	X	X	X
max-pool- size		X	X	X	X
pool-idle- timeout-in- seconds		X	X	X	X

Commit Option

The commit option controls the action taken by the EJB container when an EJB component completes a transaction. The commit option has a significant impact on performance.

The following are the possible values for the commit option:

- **Commit option B:** When a transaction completes, the bean is kept in the cache and retains its identity. The next invocation for the same primary key can use the cached instance. The EJB container will call the bean's `ejbLoad()` method before the method invocation to synchronize with the database.
- **Commit option C:** When a transaction completes, the EJB container calls the bean's `ejbPassivate()` method, the bean is disassociated from its primary key and returned to the free pool. The next invocation for the same primary key will have to get a free bean from the pool, set the `PrimaryKey` on this instance, and then call `ejbActivate()` on the instance. Again, the EJB container will call the bean's `ejbLoad()` before the method invocation to synchronize with the database.

Option B avoids `ejbActivate()` and `ejbPassivate()` calls. So, in most cases it performs better than option C since it avoids some overhead in acquiring and releasing objects back to pool.

However, there are some cases where option C can provide better performance. If the beans in the cache are rarely reused and if beans are constantly added to the cache, then it makes no sense to cache beans. With option C is used, the container puts beans back into the pool (instead of caching them) after method invocation or on transaction completion. This option reuses instances better and reduces the number of live objects in the JVM, speeding garbage collection.

Determining the Best Commit Option

To determine whether to use commit option B or commit option C, first take a look at the cache-hits value using the monitoring command for the bean. If the cache hits are much higher than cache misses, then option B is an appropriate choice. You might still have to change the `max-cache-size` and `cache-resize-quantity` to get the best result.

If the cache hits are too low and cache misses are very high, then the application is not reusing the bean instances and hence increasing the cache size (using `max-cache-size`) will not help (assuming that the access pattern remains the same). In this case you might use commit option C. If there is no great difference between cache-hits and cache-misses then tune `max-cache-size`, and probably `cache-idle-timeout-in-seconds`.

Java Message Service Settings

The Type attribute that determines whether the Java Message Service (JMS) is on local or remote system affects performance. Local JMS performance is better than remote JMS performance. However, a remote cluster can provide failover capabilities and can be administrated together, so there may be other advantages of using remote JMS. For more information on using JMS, see "Administering the Java Message Service (JMS) ([../doc.312/e24928/jms.htm#GSADG00020](http://doc.312/e24928/jms.htm#GSADG00020))" in *Oracle GlassFish Server Administration Guide*.

Transaction Service Settings

The transaction manager makes it possible to commit and roll back distributed transactions.

A distributed transactional system writes transactional activity into transaction logs so that they can be recovered later. But writing transactional logs has some performance penalty.

The following topics are addressed here:

- Monitoring the Transaction Service (#abeen)
- Tuning the Transaction Service (#abeep)

Monitoring the Transaction Service

Transaction Manager monitoring is disabled by default. Enable monitoring of the transaction service through the GlassFish Server Administration Console by navigating to the Configurations>*configuration-name*>Monitoring node. Refer to the Administration Console for complete instructions.

You can also enable monitoring with these commands:

```
set serverInstance.transaction-service.monitoringEnabled=true reconfig
serverInstance
```

Viewing Monitoring Information

To view monitoring information for the transaction service in the GlassFish Server Administration Console, navigate to the server (Admin Server) node and then select the Monitor tab.

The following statistics are gathered on the transaction service:

- **total-tx-completed** Completed transactions.
- **total-tx-rolled-back** Total rolled back transactions.
- **total-tx-inflight** Total inflight (active) transactions.
- **isFrozen** Whether transaction system is frozen (true or false)
- **inflight-tx** List of inflight (active) transactions.

Tuning the Transaction Service

This property can be used to disable the transaction logging, where the performance is of utmost importance more than the recovery. This property, by default, won't exist in the server configuration.

Most Transaction Service tuning tasks can be performed through the GlassFish Server Administration Console by navigating to the Configurations>*configuration-name*>Transaction Service node and then following the instructions in the online help. Alternatively, you can follow the instructions in "Administering Transactions ([../doc.312/e24928/transactions.htm#GSADG00022](#))" in *Oracle GlassFish Server Administration Guide*.

Disable Distributed Transaction Logging

You can disable transaction logging through the Administration Console or by using the following `asadmin set` subcommand:

```
asadmin set server1.transaction-service.disable-distributed-transaction-logging=true
```

Disabling transaction logging can improve performance. Setting it to false (the default), makes the transaction service write transactional activity to transaction logs so that transactions can be recovered. If Recover on Restart is checked, this property is ignored.

Set this property to true only if performance is more important than transaction recovery.

Recover On Restart (Automatic Recovery)

You can set the Recover on Restart attribute through the Administration Console or by entering the following `asadmin set` subcommand:

```
asadmin set server1.transaction-service.automatic-recovery=false
```

When Recover on Restart is true, the server will always perform transaction logging, regardless of the Disable Distributed Transaction Logging attribute.

If Recover on Restart is false, then:

- If Disable Distributed Transaction Logging is false (the default), then the server will write transaction logs.
- If Disable Distributed Transaction Logging is true, then the server will not write transaction logs.

Not writing transaction logs will give approximately twenty percent improvement in performance, but at the cost of not being able to recover from any interrupted transactions. The performance benefit applies to transaction-intensive tests. Gains in real applications may be less.

Keypoint Interval

The keypoint interval determines how often entries for completed transactions are removed from the log file. Keypointing prevents a process log from growing indefinitely.

Frequent keypointing is detrimental to performance. The default value of the Keypoint Interval is 2048, which is sufficient in most cases.

HTTP Service Settings

Tuning the monitoring and access logging settings for the HTTP server instances that handle client requests are important parts of ensuring peak GlassFish Server performance.

The following topics are addressed here:

- Monitoring the HTTP Service (#abeeu)
- HTTP Service Access Logging (#abefk)

Monitoring the HTTP Service

Disabling the collection of monitoring statistics can increase overall GlassFish Server performance. You can enable or disable monitoring statistics collection for the HTTP service using either the Administration Console or `asadmin` subcommands.

Refer to "Administering the Monitoring Service ([../doc.312/e24928/monitoring.htm#GSADG00011](#))" in *Oracle GlassFish Server Administration Guide* for complete instructions on configuring the monitoring service using `asadmin` subcommands.

If using the Administration Console, click the Configurations>*configuration-name*>Monitoring node for the configuration for which you want to enable or disable monitoring for selected components. Refer to the Administration Console online help for complete instructions.

For instructions on viewing comprehensive monitoring statistics using `asadmin` subcommands, see "Viewing Comprehensive Monitoring Data ([../doc.312/e24928/monitoring.htm#GSADG00560](#))" in *Oracle GlassFish Server Administration Guide*. If using the Administration Console, you can view monitoring statistics by navigating to the server (Admin Server) node, and then clicking the Monitor tab. Refer to the online help for configuring different views of the available monitoring statistics.

When viewing monitoring statistics, some key performance-related information to review includes the following:

- DNS Cache Information (dns) (#abeew)
- File Cache Information (file-cache) (#abefh)
- Keep Alive (keep-alive) (#abefi)
- Connection Queue (#abefg)

DNS Cache Information (dns)

The DNS cache caches IP addresses and DNS names. The DNS cache is disabled by default. In the DNS Statistics for Process ID All page under Monitor in the web-based Administration interface the following statistics are displayed:

- Enabled (#abeex)
- CacheEntries (CurrentCacheEntries / MaxCacheEntries) (#abeey)
- HitRatio (#abeez)
- Caching DNS Entries (#abefa)
- Limit DNS Lookups to Asynchronous (#abefb)
- Enabled (#abefc)
- NameLookups (#abefd)
- AddrLookups (#abefe)
- LookupsInProgress (#abeff)

Enabled

If the DNS cache is disabled, the rest of this section is not displayed.

By default, the DNS cache is off. Enable DNS caching in the Administration Console by clicking the Configurations>*configuration-name*>Network Config>*http-listener-name* node. Click the HTTP tab and enable the DNS Lookup option.

CacheEntries (CurrentCacheEntries / MaxCacheEntries)

The number of current cache entries and the maximum number of cache entries. A single cache entry represents a single IP address or DNS name lookup. Make the cache as large as the maximum number of clients that access your web site concurrently. Note that setting the cache size too high is a waste of memory and degrades performance.

Set the maximum size of the DNS cache by entering or changing the value in the in the Administration Console by clicking the Configurations>*configuration-name*>Network Config>*http-listener-name* node. Click the File tab and set the desired options.

HitRatio

The hit ratio is the number of cache hits divided by the number of cache lookups.

This setting is not tunable.

Note:

If you turn off DNS lookups on your server, host name restrictions will not work and IP addresses will appear instead of host names in log files.

Caching DNS Entries

It is possible to also specify whether to cache the DNS entries. If you enable the DNS cache, the server can store hostname information after receiving it. If the server needs information about the client in the future, the information is cached and available without further querying. specify the size of the DNS cache

and an expiration time for DNS cache entries. The DNS cache can contain 32 to 32768 entries; the default value is 1024. Values for the time it takes for a cache entry to expire can range from 1 second to 1 year specified in seconds; the default value is 1200 seconds (20 minutes).

Limit DNS Lookups to Asynchronous

Do not use DNS lookups in server processes because they are resource-intensive. If you must include DNS lookups, make them asynchronous.

Enabled

If asynchronous DNS is disabled, the rest of this section will not be displayed.

NameLookups

The number of name lookups (DNS name to IP address) that have been done since the server was started. This setting is not tunable.

AddrLookups

The number of address loops (IP address to DNS name) that have been done since the server was started. This setting is not tunable.

LookupsInProgress

The current number of lookups in progress.

File Cache Information (file-cache)

The file cache caches static content so that the server handles requests for static content quickly. The file-cache section provides statistics on how your file cache is being used.

For information about tuning the file cache, see [File Cache Settings \(#gkxit\)](#).

The Monitoring page lists the following file cache statistics:

- Number of Hits on Cached File Content
- Number of Cache Entries
- Number of Hits on Cached File Info
- Heap Space Used for Cache
- Number of Misses on Cached File Content
- Cache Lookup Misses
- Number of Misses on Cached File Content
- Max Age of a Cache Entry

- Max Number of Cache Entries
- Max Number of Open Entries
- Is File Cached Enabled?
- Maximum Memory Map to be Used for Cache
- Memory Map Used for cache
- Cache Lookup Hits
- Open Cache Entries: The number of current cache entries and the maximum number of cache entries are both displayed. A single cache entry represents a single URI. This is a tunable setting.
- Maximum Heap Space to be Used for Cache

Keep Alive (keep-alive)

The following are statistics related to the Keep Alive system. The most important settings you can tune here relate to HTTP Timeout. See [Timeout \(#abefu\)](#) for more information.

- Connections Terminated Due to Client Connection Timed Out
- Max Connection Allowed in Keep-alive
- Number of Hits
- Connections in Keep-alive Mode
- Connections not Handed to Keep-alive Thread Due to too Many Persistent Connections
- The Time in Seconds Before Idle Connections are Closed
- Connections Closed Due to Max Keep-alive Being Exceeded

Connection Queue

- **Total Connections Queued:** Total connections queued is the total number of times a connection has been queued. This includes newly accepted connections and connections from the keep-alive system.
- **Average Queuing Delay:** Average queueing delay is the average amount of time a connection spends in the connection queue. This represents the delay between when a request connection is accepted by the server, and a request processing thread (also known as a session) begins servicing the request.

HTTP Service Access Logging

Accessing Logging can be tuned using several `asadmin` subcommands. Refer to "Administering the Monitoring Service ([../doc.312/e24928/monitoring.htm#GSADG00011](#))" in *Oracle GlassFish Server Administration Guide* for information about using these subcommands.

If using the Administration Console, Access Logging is configured from the Configurations>*configuration-name*>HTTP Service page. Refer to the Administration Console online help for complete instructions about the options on this page.

To enable or disable access logging, check or uncheck the Access Logging Enabled checkbox. Access Logging is disabled by default.

When performing benchmarking, ensure that Access Logging is disabled. If Access Logging is enabled, it is recommended that you also enable Rotation to ensure that the logs do not run out of disk space.

Network Listener Settings

You can tune Network Listener settings from the command line by using the instructions in "Administering HTTP Network Listeners ([../doc.312/e24928/http_https.htm#GSADG00588](#))" in *Oracle GlassFish Server Administration Guide*.

If using the Administration Console, navigate to the Configurations >*configuration-name*>Network Config>Network Listeners>*listener-name* node, and then click the tab for the desired configuration page. Refer to the online help for complete instructions about the options on these tabs.

GlassFish Server Network Listener performance can be enhanced by modifying settings on the following Edit Network Listener tabs in the Administration Console:

- General Settings (#abegl)
- HTTP Settings (#gkxjd)
- File Cache Settings (#gkxit)

General Settings

For machines with only one network interface card (NIC), set the network address to the IP address of the machine (for example, 192.18.80.23 instead of default 0.0.0.0). If you specify an IP address other than 0.0.0.0, the server will make one less system call per connection. Specify an IP address other than 0.0.0.0 for best possible performance. If the server has multiple NIC cards then create multiple listeners for each NIC.

HTTP Settings

The following settings on the HTTP tab can significantly affect performance:

- Max Connections (#abeft)
- DNS Lookup Enabled (#abegd)
- Timeout (#abefu)
- Header Buffer Length (#abefq)

Max Connections

Max Connections controls the number of requests that a particular client can make over a keep-alive connection. The range is any positive integer, and the default is 256.

Adjust this value based on the number of requests a typical client makes in your application. For best performance specify quite a large number, allowing clients to make many requests.

The number of connections specified by Max Connections is divided equally among the keep alive threads. If Max Connections is not equally divisible by Thread Count, the server can allow slightly more than Max Connections simultaneous keep alive connections.

DNS Lookup Enabled

This setting specifies whether the server performs DNS (domain name service) lookups on clients that access the server. When DNS lookup is not enabled, when a client connects, the server knows the client's IP address but not its host name (for example, it knows the client as 198.95.251.30, rather than `www.xyz.com`). When DNS lookup is enabled, the server will resolve the client's IP address into a host name for operations like access control, common gateway interface (CGI) programs, error reporting, and access logging.

If the server responds to many requests per day, reduce the load on the DNS or NIS (Network Information System) server by disabling DNS lookup. Enabling DNS lookup will increase the latency and load on the system, so modify this setting with caution.

Timeout

Timeout determines the maximum time (in seconds) that the server holds open an HTTP keep alive connection. A client can keep a connection to the server open so that multiple requests to one server can be serviced by a single network connection. Since the number of open connections that the server can handle is limited, a high number of open connections will prevent new clients from connecting.

The default time out value is 30 seconds. Thus, by default, the server will close the connection if idle for more than 30 seconds. The maximum value for this parameter is 300 seconds (5 minutes).

The proper value for this parameter depends upon how much time is expected to elapse between requests from a given client. For example, if clients are expected to make requests frequently then, set the parameter to a high value; likewise, if clients are expected to make requests rarely, then set it to a low value.

Both HTTP 1.0 and HTTP 1.1 support the ability to send multiple requests across a single HTTP session. A server can receive hundreds of new HTTP requests per second. If every request was allowed to keep the connection open indefinitely, the server could become overloaded with connections. On Unix/Linux

systems, this could easily lead to a file table overflow.

The GlassFish Server's Keep Alive system, which is affected by the Timeout setting, addresses this problem. A waiting *keep alive* connection has completed processing the previous request, and is waiting for a new request to arrive on the same connection. The server maintains a counter for the maximum number of waiting keep-alive connections. If the server has more than the maximum waiting connections open when a new connection waits for a keep-alive request, the server closes the oldest connection. This algorithm limits the number of open waiting keep-alive connections.

If your system has extra CPU cycles, incrementally increase the keep alive settings and monitor performance after each increase. When performance saturates (stops improving), then stop increasing the settings.

Header Buffer Length

The size (in bytes) of the buffer used by each of the request processing threads for reading the request data from the client.

Adjust the value based on the actual request size and observe the impact on performance. In most cases the default should suffice. If the request size is large, increase this parameter.

File Cache Settings

The GlassFish Server uses a file cache to serve static information faster. The file cache contains information about static files such as HTML, CSS, image, or text files. Enabling the HTTP file cache will improve performance of applications that contain static files.

The following settings on the File Cache tab can significantly affect performance:

- Max File Count (#abegf)
- Max Age (#abegh)

Max File Count

Max File Count determines how many files are in the cache. If the value is too big, the server caches little-needed files, which wastes memory. If the value is too small, the benefit of caching is lost. Try different values of this attribute to find the optimal solution for specific applications—generally, the effects will not be great.

Max Age

This parameter controls how long cached information is used after a file has been cached. An entry older than the maximum age is replaced by a new entry for the same file.

If your Web site's content changes infrequently, increase this value for improved performance. Set the maximum age by entering or changing the value in the Maximum Age field of the File Cache Configuration page in the web-based Admin Console for the HTTP server node and selecting the File Caching Tab.

Set the maximum age based on whether the content is updated (existing files are modified) on a regular schedule or not. For example, if content is updated four times a day at regular intervals, you could set the maximum age to 21600 seconds (6 hours). Otherwise, consider setting the maximum age to the longest time you are willing to serve the previous version of a content file after the file has been modified.

Transport Settings

The Acceptor Threads property for the Transport service specifies how many threads you want in accept mode on a listen socket at any time. It is a good practice to set this to less than or equal to the number of CPUs in your system.

In the GlassFish Server, acceptor threads on an HTTP Listener accept connections and put them onto a connection queue. Session threads then pick up connections from the queue and service the requests. The server posts more session threads if required at the end of the request.

See "Administering HTTP Network Listeners ([../doc.312/e24928/http_https.htm#GSADG00771](#))" in *Oracle GlassFish Server Administration Guide* for instructions on modifying the Acceptor Threads property. If using the Administration Console, the Acceptor Threads property is available on the Configurations>*configuration-name*>Network Config>Transports>tcp page.

Thread Pool Settings

You can tune thread pool settings by following the instructions in "Administering Thread Pools ([../doc.312/e24928/threadpools.htm#GSADG00008](#))" in *Oracle GlassFish Server Administration Guide*. If using the Administration Console Thread Pool settings are available on the Configurations>*configuration-name*>Thread Pools>*thread-pool-name* page.

The following thread pool settings can have significant effects on GlassFish Server performance:

- Max Thread Pool Size (#abefn)
- Min Thread Pool Size (#abefo)

Max Thread Pool Size

The Max Thread Pool Size parameter specifies the maximum number of simultaneous requests the server can handle. The default value is 5. When the server has reached the limit or request threads, it defers processing new requests until the number of active requests drops below the maximum amount. Increasing this value will reduce HTTP response latency times.

In practice, clients frequently connect to the server and then do not complete their requests. In these cases, the server waits a length of time specified by the Timeout parameter.

Also, some sites do heavyweight transactions that take minutes to complete. Both of these factors add to the maximum simultaneous requests that are required. If your site is processing many requests that take many seconds, you might need to increase the number of maximum simultaneous requests.

Adjust the thread count value based on your load and the length of time for an average request. In general, increase this number if you have idle CPU time and requests that are pending; decrease it if the CPU becomes overloaded. If you have many HTTP 1.0 clients (or HTTP 1.1 clients that disconnect frequently), adjust the timeout value to reduce the time a connection is kept open.

Suitable Request Max Thread Pool Size values range from 100 to 500, depending on the load. If your system has extra CPU cycles, keep incrementally increasing thread count and monitor performance after each incremental increase. When performance saturates (stops improving), then stop increasing thread count.

Min Thread Pool Size

The Min Thread Pool Size property specifies the minimum number of threads the server initiates upon startup. The default value is 2. Min Thread Pool Size represents a hard limit for the maximum number of active threads that can run simultaneously, which can become a bottleneck for performance.

Specifying the same value for minimum and maximum threads allows GlassFish Server to use a slightly more optimized thread pool. This configuration should be considered unless the load on the server varies quite significantly.

ORB Settings

The GlassFish Server includes a high performance and scalable CORBA Object Request Broker (ORB). The ORB is the foundation of the EJB Container on the server.

For complete instructions on configuring ORB settings, refer to "Administering the Object Request Broker (ORB) ([../doc.312/e24928/orb.htm#GSADG00018](#))" in *Oracle GlassFish Server Administration Guide*. Also refer to "RMI-IIOP Load Balancing and Failover ([../doc.312/e24934/rmi-iiop.htm#GSHAG00013](#))" in *Oracle GlassFish Server High Availability Administration Guide*. You can also configure most ORB settings through the GlassFish Server Administration Console by navigating to the Configurations>*configuration-name*> ORB node and then following the instructions in the Administration Console online help.

The following topics are addressed here:

- Overview (#abegu)
- How a Client Connects to the ORB (#abegv)
- Monitoring the ORB (#abegw)
- Tuning the ORB (#abegz)

Overview

The ORB is primarily used by EJB components via:

- RMI-IIOP path from an application client (or rich client) using the application client container.

- RMI/IIOP path from another GlassFish Server instance ORB.
- RMI/IIOP path from another vendor's ORB.
- In-process path from the Web Container or MDB (message driven beans) container.

When a server instance makes a connection to another server instance ORB, the first instance acts as a client ORB. SSL over IIOP uses a fast optimized transport with high-performance native implementations of cryptography algorithms.

It is important to remember that EJB local interfaces do not use the ORB. Using a local interface passes all arguments by reference and does not require copying any objects.

How a Client Connects to the ORB

A rich client Java program performs a new `initialContext()` call which creates a client side ORB instance. This in turn creates a socket connection to the GlassFish Server IIOP port. The reader thread is started on the server ORB to service IIOP requests from this client. Using the `initialContext`, the client code does a lookup of an EJB deployed on the server. An IOR which is a remote reference to the deployed EJB on the server is returned to the client. Using this object reference, the client code invokes remote methods on the EJB.

`InitialContext` lookup for the bean and the method invocations translate the marshalling application request data in Java into IIOP message(s) that are sent on the socket connection that was created earlier on to the server ORB. The server then creates a response and sends it back on the same connection. This data in the response is then un-marshalled by the client ORB and given back to the client code for processing. The Client ORB shuts down and closes the connection when the rich client application exits.

Monitoring the ORB

ORB statistics are disabled by default. To gather ORB statistics, enable monitoring with the following `asadmin` command:

```
set serverInstance.iiop-service.orb.system.monitoringEnabled=true reconfig
serverInstance
```

If using the Administration Console, you can enable ORB monitoring on the Configurations>*configuration-name*>Monitoring page. To view ORB monitoring statistics through the Administration Console, navigate to the server (Admin Server) page and click the Monitor tab. Refer to the Administration Console online help for complete instructions.

The following statistics are of particular interest when tuning the ORB:

- Connection Statistics (#abegx)
- Thread Pools (#abegy)

Connection Statistics

The following statistics are gathered on ORB connections:

- **total-inbound-connections** : Total inbound connections to ORB.
- **total-outbound-connections** : Total outbound connections from ORB.

Use the following command to get ORB connection statistics:

```
asadmin get --monitor serverInstance.iiop-service.orb.system.orb-connection.*
```

Thread Pools

The following statistics are gathered on ORB thread pools:

- **thread-pool-size** : Number of threads in ORB thread pool.
- **waiting-thread-count** : Number of thread pool threads waiting for work to arrive.

Use the following command to display ORB thread pool statistics:

```
asadmin get --monitor serverInstance.iiop-service.orb.system.orb-thread-pool.*
```

Tuning the ORB

Tune ORB performance by setting ORB parameters and ORB thread pool parameters. You can often decrease response time by leveraging load-balancing, multiple shared connections, thread pool and message fragment size. You can improve scalability by load balancing between multiple ORB servers from the client, and tuning the number of connection between the client and the server.

The following topics are addressed here:

- Tunable ORB Parameters (#abeha)
- ORB Thread Pool Parameters (#abehb)
- Client ORB Properties (#abehc)
- Thread Pool Sizing (#abehg)
- Examining IIOP Messages (#abehh)

Tunable ORB Parameters

You can tune ORB parameters using the instructions in "Administering the Object Request Broker (ORB) ([../doc.312/e24928/orb.htm#GSADG00018](#)) " in *Oracle GlassFish Server Administration Guide*. If using the Administration Console, navigate to the Configurations>*configuration-name*>ORB node and refer to the online help.

The following table summarizes the tunable ORB parameters.

Table 3-2 Tunable ORB Parameters

Path	ORB Modules	Server Settings
RMI/ IIOP from application client to application server	communication infrastructure, thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds
RMI/ IIOP from ORB to GlassFish Server	communication infrastructure, thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds
RMI/ IIOP from a vendor ORB	parts of communication infrastructure, thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds
In-process	thread pool	steady-thread-pool-size, max-thread-pool-size, idle-thread-timeout-in-seconds

ORB Thread Pool Parameters

The ORB thread pool contains a task queue and a pool of threads. Tasks or jobs are inserted into the task queue and free threads pick tasks from this queue for execution. Do not set a thread pool size such that the task queue is always empty. It is normal for a large application's Max Pool Size to be ten times the size of the current task queue.

The GlassFish Server uses the ORB thread pool to:

- Execute every ORB request
- Trim EJB pools and caches
- Execute MDB requests

Thus, even when one is not using ORB for remote-calls (via RMI/ IIOP), set the size of the threadpool to facilitate cleaning up the EJB pools and caches.

You can set ORB thread pool attributes using the instructions in "Administering Thread Pools ([../doc.312/e24928/threadpools.htm#GSADG00008](#)) " in *Oracle GlassFish Server Administration Guide*. If using the Administration Console, navigate to Configurations>*configuration-name*> Thread Pools>*thread-pool-name*, where *thread-pool-name* is the thread pool ID selected for the ORB. Thread pools have the following attributes that affect performance.

- Minimum Pool Size: The minimum number of threads in the ORB thread pool. Set to the average number of threads needed at a steady (RMI/ IIOP) load.
- Maximum Pool Size: The maximum number of threads in the ORB thread pool.
- Idle Timeout: Number of seconds to wait before removing an idle thread from pool. Allows shrinking of the thread pool.
- Number of Work Queues

In particular, the maximum pool size is important to performance. For more information, see Thread Pool Sizing ([#abehg](#)) .

Client ORB Properties

Specify the following properties as command-line arguments when launching the client program. You do this by using the following syntax when starting the Java VM:

```
-Dproperty=value
```

The following topics are addressed here:

- Controlling Connections Between Client and Server ORB ([#abehd](#))
- Load Balancing ([#abehf](#))

Controlling Connections Between Client and Server ORB

When using the default JDK ORB on the client, a connection is established from the client ORB to the application server ORB every time an initial context is created. To pool or share these connections when they are opened from the same process by adding to the configuration on the client ORB.

```
-Djava.naming.factory.initial=com.sun.enterprise.naming.SerialInitContextFactory
```

Load Balancing

For information on how to configure HTTP load balancing, including the Load Balancer Plug-in, see "Configuring Web Servers for HTTP Load Balancing (../doc.312/e24934/web-servers-for-http-load-balancing.htm#GSHAG00008) " and "Configuring HTTP Load Balancing (../doc.312/e24934/http-load-balancing.htm#GSHAG00009) " in *Oracle GlassFish Server High Availability Administration Guide*.

For information on how to configure RMI/IOP for multiple application server instances in a cluster, "RMI-IOP Load Balancing and Failover (../doc.312/e24934/rmi-iop.htm#GSHAG00013) " in *Oracle GlassFish Server High Availability Administration Guide*.

When tuning the client ORB for load-balancing and connections, consider the number of connections opened on the server ORB. Start from a low number of connections and then increase it to observe any performance benefits. A connection to the server translates to an ORB thread reading actively from the connection (these threads are not pooled, but exist currently for the lifetime of the connection).

Thread Pool Sizing

After examining the number of inbound and outbound connections as explained above, tune the size of the thread pool appropriately. This can affect performance and response times significantly.

The size computation takes into account the number of client requests to be processed concurrently, the resource (number of CPUs and amount of memory) available on the machine and the response times required for processing the client requests.

Setting the size to a very small value can affect the ability of the server to process requests concurrently, thus affecting the response times since requests will sit longer in the task queue. On the other hand, having a large number of worker threads to service requests can also be detrimental because they consume system resources, which increases concurrency. This can mean that threads take longer to acquire shared structures in the EJB container, thus affecting response times.

The worker thread pool is also used for the EJB container's housekeeping activity such as trimming the pools and caches. This activity needs to be accounted for also when determining the size. Having too many ORB worker threads is detrimental for performance since the server has to maintain all these threads. The idle threads are destroyed after the idle thread timeout period.

Examining IOP Messages

It is sometimes useful to examine the IOP messages passed by the GlassFish Server. To make the server save IOP messages to the `server.log` file, set the JVM option `-Dcom.sun.CORBA.ORBDebug=giop` . Use the same option on the client ORB.

The following is an example of IOP messages saved to the server log. Note: in the actual output, each line is preceded by the timestamp, such as `[29/Aug/2002:22:41:43] INFO (27179): CORE3282: stdout` .

```
+++++ Message(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): createFromStream: type is 4 < MessageBase(Thread[ORB
Client-side Reader, conn to 192.18.80.118:1050,5,main]): Message GIOP version:
1.2 MessageBase(Thread[ORB Client-side Reader, conn to
192.18.80.118:1050,5,main]): ORB Max GIOP Version: 1.2 Message(Thread[ORB Client-
side Reader, conn to 192.18.80.118:1050,5,main]): createFromStream: message
construction complete. com.sun.corba.ee.internal.iiop.MessageMediator (Thread[ORB
Client-side Reader, conn to 192.18.80.118:1050,5,main]): Received message: -----
Input Buffer ----- Current index: 0 Total length : 340 47 49 4f 50 01 02 00 04 0
0 00 01 48 00 00 00 05 GIOP.....H....
```

Note:

The flag `-Dcom.sun.CORBA.ORBdebug=giop` generates many debug messages in the logs. This is used only when you suspect message fragmentation.

In this sample output above, the `createFromStream` type is shown as `4`. This implies that the message is a fragment of a bigger message. To avoid fragmented messages, increase the fragment size. Larger fragments mean that messages are sent as one unit and not as fragments, saving the overhead of multiple messages and corresponding processing at the receiving end to piece the messages together.

If most messages being sent in the application are fragmented, increasing the fragment size is likely to improve efficiency. On the other hand, if only a few messages are fragmented, it might be more efficient to have a lower fragment size that requires smaller buffers for writing messages.

Resource Settings

Tuning JDBC and connector resources can significantly improve GlassFish Server performance.

The following topics are addressed here:

- [JDBC Connection Pool Settings \(#abehq\)](#)
- [Connector Connection Pool Settings \(#abeby\)](#)

JDBC Connection Pool Settings

For optimum performance of database-intensive applications, tune the JDBC Connection Pools managed by the GlassFish Server. These connection pools maintain numerous live database connections that can be reused to reduce the overhead of opening and closing database connections. This section describes how to tune JDBC Connection Pools to improve performance.

J2EE applications use JDBC Resources to obtain connections that are maintained by the JDBC Connection Pool. More than one JDBC Resource is allowed to refer to the same JDBC Connection Pool. In such a case, the physical connection pool is shared by all the resources.

Refer to "Administering Database Connectivity (../doc.312/e24928/jdbc.htm#GSADG00015)" in *Oracle GlassFish Server Administration Guide* for more information about managing JDBC connection pools.

The following topics are addressed here:

- Monitoring JDBC Connection Pools (#abehr)
- Tuning JDBC Connection Pools (#abebs)

Monitoring JDBC Connection Pools

Statistics-gathering is disabled by default for JDBC Connection Pools. Refer to for instructions on enabling JDBC monitoring in "Administering the Monitoring Service (../doc.312/e24928/monitoring.htm#GSADG00011)" in *Oracle GlassFish Server Administration Guide*. If using the Administration Console, JDBC monitoring can be enabled on the Configurations>*configuration-name*>Monitoring page.

The following attributes are monitored:

- **numConnFailedValidation (count)** Number of connections that failed validation.
- **numConnUsed (range)** Number of connections that have been used.
- **numConnFree (count)** Number of free connections in the pool.
- **numConnTimedOut (bounded range)** Number of connections in the pool that have timed out.

To get JDBC monitoring statistics, use the following commands:

```
asadmin get --monitor=true serverInstance.resources.jdbc-connection-pool.*asadmin  
get --monitor=true serverInstance.resources.jdbc-connection-pool. poolName.* *
```

To view JDBC monitoring statistics through the Administration Console, navigate to the server (Admin Server) page and click the Monitor tab. Refer to the Administration Console online help for complete instructions.

Tuning JDBC Connection Pools

Refer to "Administering Database Connectivity (../doc.312/e24928/jdbc.htm#GSADG00015)" in *Oracle GlassFish Server Administration Guide* for instructions on configuring JDBC connection pools. If using the GlassFish Server Administration Console by navigating to the Resources>JDBC>JDBC Connection Pools>*jdbc-pool-name* page and then clicking the desired tab.

The following JDBC properties affect GlassFish Server performance:

- Pool Size Settings (#abeht)
- Timeout Settings (#abehu)
- Isolation Level Settings (#abehv)
- Connection Validation Settings (#abehw)

Pool Size Settings

Pool Size settings can be configured in the Pool Settings section on the General tab in the Edit JDBC Connection Pool page.

The following settings control the size of the connection pool:

- **Initial and Mimimum Pool Size:** Size of the pool when created, and its minimum allowable size.
- **Maximum Pool Size:** Upper limit of size of the pool.
- **Pool Resize Quantity:** Number of connections to be removed when the idle timeout expires.
Connections that have idled for longer than the timeout are candidates for removal. When the pool size reaches the initial and minimum pool size, removal of connections stops.

The following table summarizes advantages and disadvantages to consider when sizing connection pools.

Table 3-3 Connection Pool Sizing

Connection Pool	Advantages	Disadvantages
Small Connection pool	Faster access on the connection table.	May not have enough connections to satisfy requests. Requests may spend more time in the queue.
Large Connection pool	More connections to fulfill requests. Requests will spend less (or no) time in the queue	Slower access on the connection table.

Timeout Settings

The following JDBC timeout settings can be configured on the in the Pool Settings section on the General tab in the Edit JDBC Connection Pool page.

- **Max Wait Time:** Amount of time the caller (the code requesting a connection) will wait before getting a connection timeout. The default is 60 seconds. A value of zero forces caller to wait indefinitely.

To improve performance set Max Wait Time to zero (0). This essentially blocks the caller thread until a connection becomes available. Also, this allows the server to alleviate the task of tracking the elapsed wait time for each request and increases performance.

- **Idle Timeout:** Maximum time in seconds that a connection can remain idle in the pool. After this time, the pool can close this connection. This property does not control connection timeouts on the database server.

Keep this timeout shorter than the database server timeout (if such timeouts are configured on the database), to prevent accumulation of unusable connection in GlassFish Server.

For best performance, set Idle Timeout to zero (0) seconds, so that idle connections will not be removed. This ensures that there is normally no penalty in creating new connections and disables the idle monitor thread. However, there is a risk that the database server will reset a connection that is unused for too long.

Isolation Level Settings

The following JDBC Isolation Level settings can be configured in the Transaction section on the General tab in the Edit JDBC Connection Pool page.

- **Transaction Isolation:** Specifies the transaction isolation level of the pooled database connections. If this parameter is unspecified, the pool uses the default isolation level provided by the JDBC Driver.
- **Isolation Level Guaranteed:** Guarantees that every connection obtained from the pool has the isolation specified for the Transaction Isolation level. Applicable only when the Transaction Isolation level is specified. The default value is Guaranteed.

This setting can have some performance impact on some JDBC drivers. Set to false when certain that the application does not change the isolation level before returning the connection.

Avoid specifying the Transaction Isolation level. If that is not possible, consider disabling the Isolation Level Guaranteed option and then make sure applications do not programmatically alter the connections; isolation level.

If you must specify a Transaction Isolation level, specify the best-performing level possible. The isolation levels listed from best performance to worst are:

1. `READ_UNCOMMITTED`
2. `READ_COMMITTED`
3. `REPEATABLE_READ`
4. `SERIALIZABLE`

Choose the isolation level that provides the best performance, yet still meets the concurrency and consistency needs of the application.

Connection Validation Settings

JDBC Connection Validation settings can be configured in the Connection Validation section on the Advanced tab in the Edit JDBC Connection Pool page.

- **Connection Validation Required:** If enabled, the pool validates connections (checks to find out if they are usable) before providing them to an application.

If possible, keep this option disabled. Requiring connection validation forces the server to apply the validation algorithm every time the pool returns a connection, which adds overhead to the latency of `getConnection()`. If the database connectivity is reliable, you can omit validation.

- **Validation Method:** Specifies the type of connection validation to perform. Must be one of the following:
 - `auto-commit` : Attempt to perform an auto-commit on the connection.
 - `metadata` : Attempt to get metadata from the connection.
 - `table` : Performing the query on a specified table. If this option is selected, Table Name must also be set. Choosing this option may be necessary if the JDBC driver caches calls to `setAutoCommit()` and `getMetaData()`.
 - `custom-validation` : Define a custom validation method.
- **Table Name:** Name of the table to query when the Validation Method is set to `table`.
- **Close All Connections On Any Failure:** Specify whether all connections in the pool should be closed if a single validation check fails. This option is disabled by default. One attempt will be made to re-establish failed connections.

Connector Connection Pool Settings

From a performance standpoint, connector connection pools are similar to JDBC connection pools. Follow all the recommendations in the previous section, [Tuning JDBC Connection Pools \(#abehs\)](#) .

Transaction Support

You may be able to improve performance by overriding the default transaction support specified for each connector connection pool.

For example, consider a case where an Enterprise Information System (EIS) has a connection factory that supports local transactions with better performance than global transactions. If a resource from this EIS needs to be mixed with a resource coming from another resource manager, the default behavior forces the use of XA transactions, leading to lower performance. However, by changing the EIS's connector connection pool to use LocalTransaction transaction support and leveraging the Last Agent Optimization feature previously described, you could leverage the better-performing EIS LocalTransaction implementation. For more information on LAO, see [Configure JDBC Resources as One-Phase Commit Resources \(tuning-apps.htm#abecq\)](#) .

You can specify transaction support when you create or edit a connector connection pool.

Also set transaction support using `asadmin` . For example, the following `asadmin` command could be used to create a connector connection pool `TESTPOOL` with `transaction-support` set to `LOCAL` .

```
asadmin> create-connector-connection-pool --raname jdbcra --connectiondefinition
javax.sql.DataSource -transactionsupport LocalTransaction TESTPOOL
```

Load Balancer Settings

Oracle GlassFish Server includes a Load Balancer Plug-in for popular Web servers such as Oracle HTTP Server, Oracle iPlanet Web Server, Apache HTTP Server, and Microsoft Windows IIS. The Load Balancer Plug-in includes a graphical Load Balancer Configurator installation wizard that makes it easy to configure the plug-in to work with your particular GlassFish Server and Web Server installations.

GlassFish Server load balancing configurations can vary widely depending on the needs of your enterprise and are beyond the scope of this Performance Tuning Guide. For complete information about configuring load balancing in GlassFish Server, refer to the following documentation:

- "Configuring Web Servers for HTTP Load Balancing ([../doc.312/e24934/web-servers-for-http-load-balancing.htm#GSHAG00008](#)) " in *Oracle GlassFish Server High Availability Administration Guide*
- "Configuring HTTP Load Balancing ([../doc.312/e24934/http-load-balancing.htm#GSHAG00009](#)) " in *Oracle GlassFish Server High Availability Administration Guide*
- "RMI-IIOP Load Balancing and Failover ([../doc.312/e24934/rmi-iiop.htm#GSHAG00013](#)) " in *Oracle GlassFish Server High Availability Administration Guide*

About Oracle (<http://www.oracle.com/corporate/index.html>) | Contact Us (<http://www.oracle.com/us/corporate/contact/index.html>) | Legal Notices



(<http://www.oracle.com/us/legal/index.html>) | Terms of Use (<http://www.oracle.com/us/legal/terms/index.html>) | Your Privacy Rights

(<http://www.oracle.com/us/legal/privacy/index.html>)

Copyright © 2015, Oracle and/or its affiliates. All rights reserved.