

Pramati Server

Performance Tuning Guide

Pramati Server 6.0 Performance Tuning Guide

ID 6000-005

April 2008

Pramati Technologies

301 White House, Begumpet

Hyderabad 500 016

India

info@pramati.com

www.pramati.com

Communications Design Group, Pramati Technologies.

Made in India.

Copyright 2007 Pramati Technologies. All rights reserved. Pramati is a registered trademark of Pramati Technologies in United States and India.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc., in United States and other countries. All other trademarks may be the registered trademarks of their respective owners.

No third party intellectual property right liability is assumed with respect to the information contained herein. While the information in this publication is believed to be accurate, Pramati Technologies makes no warranty of any kind to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

Pramati Technologies shall not be liable for errors contained herein, or for incidental or consequential damages in connection with the furnishing, performance or use of this material. Pramati Technologies assumes no responsibility for errors or omissions contained in this book. This publication and features described herein are subject to change without notice. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, photocopying, recording or otherwise, without prior written consent of Pramati Technologies.

1	Preface	.1
	About This Guide	.1
	Audience	.2
	Document Conventions	.2
	Tunable Points	.3
	Documentation Keywords	.4
	Performance Tuning Guide Roadmap	.4
	Related Documents	.5
	Contact Information	.6
2	System Configuration	.7
	JVM Tuning	.7
	Setting JVM Parameters in Startup Script	.7
	Using -client/-server options	.7
	Managing System Memory	.8
	Setting Garbage Collection	.8
	Threading Models	.9
3	Tuning Web Container	.11
	Processing Concurrent Requests using the <worker-thread-count>	.11
	Viewing Worker-threads	.12
	Acceptor-thread Count	.12
	Varying Cache Size	.13
	Dynamic Caching	.13
	Session Timeout	.14
	Default Session Timeout	.14
	Cleaning up Stale Sessions	.15
	Network Parameters	.15
	Adjusting Packet Sizes with Traffic	.15
	Releasing Sockets for Reuse	.16
	Buffer Size for Sending Data	.16
	Buffer Size for Receiving Data	.16
	Thread Funneling	.17
	Worker Thread Count	.17
	Threshold for Service Unavailable	.18
	Priority Processing	.18
	Using the <acceptor-thread-count> tag	.19
4	Optimizing Resources	.21
	Resource Properties	.21
	Max Pool Size	.21
	Min Pool Size	.21
	Initial Pool Size	.22
	Idle Timeout	.22
	Connection Request Timeout	.22
	Cached Prepared Statements	.22
	Tuning Resource Parameters From the Console	.23
5	Tuning EJB Container	.25
	Container Resource Pools	.25
	Tuning Bean Pool Sizes	.25

Max Timer Threads	26
Resource Waits	26
Pool Wait Timeout	26
Primary Key Wait Timeout (Lock Timeout)	27
Time-Based Operations	27
Session Timeout	27
6 Load Balancing in Cluster	29
Load Balancing for Web Nodes	29
Load Balancing for Java clients	29
Load Balancer Algorithm	29
Weighted Round Robin	30
Setting Load Balancing Properties	30
Customizing Load Balancer	30
Writing Applications for Load Balancing	30
Concurrency Control	32
Session beans	32
Entity beans	32
Failover Mechanism	32
Stateless Session Beans	32
Stateful Session Beans	32
Entity Beans	33
Redeployment Issues	33
7 Naming Service	35
Contextual Lookup	35
Caching on Client-Side	35
Support for Hierarchy	36
Binding Object Types	36
UserTransaction Support	36
Initial Context Factory	36
Properties for InitialContext	36
8 Monitoring and Tuning Tools	39
Server Statistics	39
Web Container Statistics	39
Resource Statistics	40
JMS Statistics	41
Server Diagnostics	42
Performance Criteria	43
Criteria for Selecting Tools to Test Performance	43
Tools and Utilities for Performance Tuning	43

IT managers want to reduce system costs and complexity while maintaining acceptable levels of performance for end users. Tuning for performance is a crucial part of development process which is often overlooked. Major performance issues arise due to contention in one or more of the following resources:

- CPU time
- Available memory
- I/O operations
- Database access

To overcome most performance related issues, it is necessary to keep the above factors in check. It may not be possible to tune an application server for all applications running on it at the same time. An optimum can be reached through characterization.

Performance characterization is required when:

- Performance is better on one application server
- Performance of the application is poor

Tuning an application server is server specific and cannot be done inside an application. Default parameters are tuned based on a best guess regarding the activities in the server. Pramati Server provides a number of tunable parameters that affect the performance. Most of the parameters can be configured by editing XML files located in the Server `config` directory or through the Pramati Server Management Console.

About This Guide

The *Pramati Server Performance Tuning Guide* outlines the methodology to profile performance characteristics of Pramati Server as a small, medium, or large enterprise business server operating under various workloads. This guide provides information on how to tune Pramati Server to match application needs. Instructions to configure different operating parameters of Pramati Server and how to tune an application are provided.

To get the best performance out of Pramati Server:

- Choose the appropriate platform. Pramati Server can be used on a variety of platforms. A list of supported platforms is listed on the Web site at <http://www.pramati.com/docstore/1210013/psv60platcert.htm>.

- Modify start-up level parameters using JVM Options best suited to your environment.
- Modify Web container parameters to suit the application needs.
- Optimize resources.
- Modify bean pools and O-R mappings.

We will review each of these in course of this book and discuss how they affect Server performance.

Audience

The *Pramati Server Performance Tuning Guide* is intended for developers who want guidelines to use Pramati Server to consolidate multiple applications, and assumes that they are familiar with Pramati Server.

Document Conventions

The following conventions are commonly used in this guide:

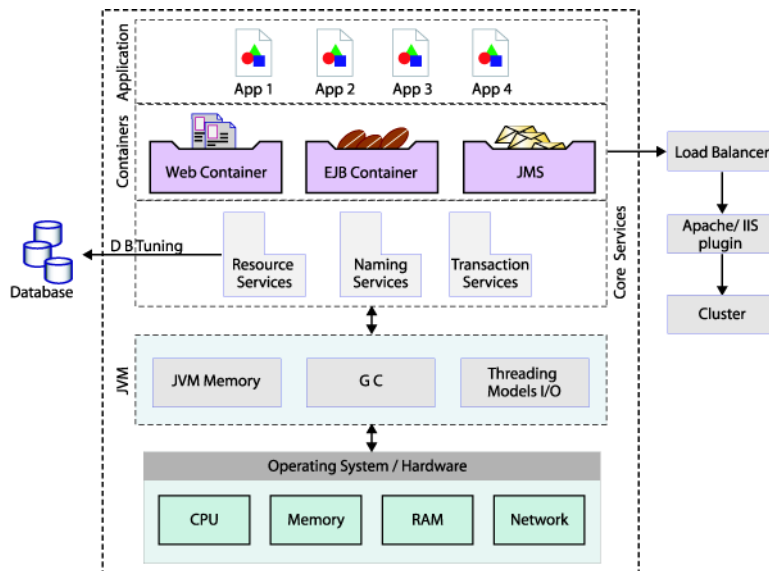
Table 1: Document conventions used in the Pramati Server Performance Tuning Guide

Convention	Refers to
Server	Pramati Server
Console	Pramati Server Management Console
Deploy Tool	Pramati Deploy Tool
Item A > Item B	Menu and sub-menu selections
code	Font used for writing code snippets
<i>Note:</i>	Indented paragraphs in italics denote notes
press Enter	Press the suggested keys using the keyboard where suggested
<code><install_dir></code>	Server installation directory

Screenshots used in this document represent the graphical user interface (GUI) of the Management Console on the Windows NT platform. The GUI has a Java look and feel and may differ according to the operating system used.

Tunable Points

There are a number of parameters that affect tuning of applications running on the application server and many areas that the system administrator can check while tuning. This involves hardware, network, and JVM settings. The following diagram shows the various tunable points in an application server:



The basic level of tunable points form the hardware and the operating system on which the server is running. The next level is the JVM. Here, the JVM memory model, the Garbage Collection, and the threading model used need to be configured. The top layer comprises of the application server and the applications themselves. This layer contains various services such as Resource service, Naming service, and Transaction service. Pramati Server enables tuning of these services through the Web-based Management Console or by manually editing the configuration files.

The Web container, EJB container, JMS service, and deployment descriptors are important points. Other tunable points may not be an integral part of the environment, but affect server performance. For example, the database servers, which may be on another machine have an impact on the performance.

When load balancing is used, entities such as Apache or IIS plugins have an impact.

Documentation Keywords

This guide uses some key terms that are defined in the table below:

Table 2: Key terminology used in Pramati Server Performance Tuning Guide

Key Term	Description
Console	The Pramati Server Management Console.
Load simulator	A program or a tool used to stress an application or server.
Java Client	Clients can access EJB applications deployed on Pramati Server through a Java Client that supports, or is compatible with Remote Method Invocation (RMI) specification. These applications can also be accessed by making an HTTP request through a Web browser.
Nagle's algorithm	Reduces the network traffic by optimizing the packet size being transferred over the network.
Concurrency-level	The level at which one or more users operate at the same time.
Prepared statements	A SQL statement is precompiled and stored in a PreparedStatement object which can be used to efficiently execute this statement multiple times.
Activation	The process in which an object is brought back into the system.
Passivation	The process in which an object is taken out of the system, but not destroyed.
Load Balancing	It is the algorithm used in clustering to distribute a request over as many machines as possible. The implementation of this algorithm decides the amount of load each node in the cluster handles.
Round Robin Load balancer	Routes user requests to different EJB nodes in a round robin fashion.
Weighted Round Robin	Routes user requests to EJB nodes based on their weights. Defaults to RRLB if no weights are specified.
Look-ups	Retrieves the named object from the naming service.
Failover	Signifies no single point of failure. This implies that clients attached to a failed node seamlessly move to a running node.
Intelligent Wrappers	These contain information required for a smooth failover. They are completely transparent to the client and require no special programming on the client-side.
Class loader	An object that is responsible for loading classes.

Performance Tuning Guide Roadmap

The chapters in this guide discuss the components that can be tuned for performance, in this order:

- **Tuning the System Configuration**

This chapter discusses how to tune the Server runtime environment and JVM options. Hardware and software play an important role in performance of an application.

- **Web Container**

This chapter discusses how to tune a Web container for performance. Web container performance can be tuned done using the Console or options provided in the `web-config.xml` and the `default-web.xml` file. Changes made using the Console are effective at runtime.

- **Resource Service**

The chapter discusses the `resource-config.xml` file and explains various tunable parameters for JDBC resources. Any deployed application requires resources such as database connectivity which can be optimized for better performance.

- **EJB Container**

This chapter discusses tuning parameters for EJB container. The application server makes EJBs available in a pool and serves them to a request that requires a specific bean. Applications can be tuned by modifying the number of beans available to a application, depending on the activity of the application and frequency of request for the bean. This can be managed using `pramati-j2ee-server.xml`.

- **Naming Service**

This chapter discusses how to tune the Naming services.

- **Load Balancing**

This chapter discusses tuning of the load balancer by setting load balancing properties using the following tag in `web-lbconfig.xml`:

```
<node-chooser  
class="com.pramati.web.lb.nodechooser.LoadBalancingNodeChooser"  
enabled="true" name="load-balancing-node-chooser"/>
```

You can add our own node-chooser class here - either along with this or inplace of this. This default `com.pramati.web.lb.nodechooser.LoadBalancingNodeChooser` class is a round robbin node chooser. The logic can be customised and plugged in here as another node-chooser entry.

- **Monitoring and Tuning Tools**

This chapter discusses various tools available for monitoring activity on the Server. The Web-based Console that is built-in Pramati Server, provides information on how the resources are being utilized and enables performance tuning.

Related Documents

To get more information about Pramati Server, refer to the following Server documentation:

- Pramati Server Installation and Configuration Guide
- Pramati Server Administration Guide
- Pramati Server Deployment Guide
- Pramati Server Security Guide
- Pramati Server Technical Reference Guide

- Server Online Help
- Server API Javadoc

Contact Information

Detailed information on this release of the product is given in the Release Notes. For additional help with using the Server and troubleshooting, visit <http://www.pramati.com> or visit <http://www.pramati.com/DevPortalWeb/forum/>.

You can also write in to support@pramati.com for your troubleshooting queries.

Performance of an application server directly depends on the environment in which it is running. Pramati Server has been certified on various platforms. Recent updates of supported platforms is available in the *Release Notes* listed on the Web site at <http://www.pramati.com/docstore/1210013/psv60relnotes.htm>. To tune the operating system and database for better performance, please check the related documentation of the vendor.

This chapter discusses various JVM options that can be used to tune the JVM. By using these options in the startup script (`runserver.bat/runserver.sh`), the performance of JVM can be enhanced, thus yielding faster response times.

JVM Tuning

This section discusses procedures to tune parameters in the VM such as memory, heap size, and explore the threading models.

Setting JVM Parameters in Startup Script

Pramati Server is started by running the `runserver.bat` or `runserver.sh` script located in the `<install_dir>/server/bin/` directory. Following is a snippet from the script:

```
...
java -Dinstall.root=%install_root%
-Djava.security.policy=%install_root%\lib\pramati\pramati.java.policy
-Djava.security.auth.policy=%install_root%\lib\pramati\pramati.jaas.policy
-Djacorb.home=%install_root%\lib\tp\jacorb
-Djava.endorsed.dirs=%install_root%\lib\std\endorsed com.pramati.Server %*
...
```

Using `-client`/`-server` options

The JVM provided by Sun™ ships with two different compilers. These compilers interface the same runtime system. For applications where faster startup time is required, the `-client` option is used.

The `-server` option is used when the overall performance is important.

Note: The first argument passed to the `java` command must be either `-client` or `-server`. The default option is `-client`.

Managing System Memory

Performance is affected by size of the memory and Garbage Collection (GC) mechanisms. Memory is managed in generations as memory pools holding objects of different ages. GC occurs in each generation, when the memory fills up.

Objects in Java are created using the `new` operator. The JVM allocates memory for these objects at runtime. This memory is finite and can exhaust if many objects are created. When this happens, the VM throws an `OutOfMemory` exception.

To avoid this situation and to make sure that enough memory is allocated to the VM, use the command line flags: `Xms` and `Xmx`. It is also a good idea to have the same values for both. This value can be set by deciding how much of heap space would be enough for the run time environment. Setting both the values to the same limit avoids the overhead of adding more memory to the heap at runtime.

To set optimum size for an application, run the application under a typical scenario. A real-life scenario involves:

- Occurrence of normal user load
- Occurrence of peak load which can be different for different applications

Ensure that GC occurs regularly at runtime . Set `-Xmx` at maximum possible value in the server. Determine the actual memory allocated by the VM which is the ideal maximum level. This is important as GC does not happen for some VMs until the threshold is reached.

Note: Minimum level must be 50% of the ideal maximum value.

Decide the memory size based on a continuous monitoring of the system and set it accordingly.

Setting Garbage Collection

GC occurs all the time in a VM. If it does not happen parallel to server activity, optimum performance is not achieved. Some VMs have an option of performing concurrent GC. If this option is enabled, Pramati Server provides high performance. Also, each VM has a set of parameters that can be tuned to enable high performance. For example, one parameter is `Nursery size`. To set the value, the VM must enable `verbosegc`. If this is set at server runtime, the VM prints messages as and when GC occurs.

Check for GC pattern and check the requirement of the application such as amount of memory freed and frequency at which memory is freed. Based on this, set other GC options for the VM.

Threading Models

Selecting the right threading model is important as each job is done by a thread that needs a lot of system resources. Available threads are native threads, pure Java threads, and other VM specific threading models.

For example, consider a many-to-one case where multiple Java threads map to one native thread. If one Java thread is waiting for an I/O, the remaining threads also wait as that native thread is also waiting for the I/O.

I/O Model

I/O is one of the most resource consuming operation. Solaris provides an option: -
`Xconcurrentio` for better performances. In this case, many-to-one threading model brings out an optimum performance.

The Web container provides tunable parameters that can improve the performance of Server and applications running on it. The Console provides a user-friendly way to modify the parameters, at runtime. Changes are ultimately affected in the `web-config.xml` file located in `<install_dir>/server/nodes/<node-name>/config` folder. The `<web-container>` tag in the `web-config.xml` contains parameters that can be modified.

This chapter discusses tuning the Web container using the Web-based Console. All tunable parameters can be entered in the `<performance-tuning-properties>` tag in the `web-config.xml` file.

Processing Concurrent Requests using the `<worker-thread-count>`

The worker-thread count is the number of requests that can be processed concurrently by the Web server. The worker-thread count for the server must be increased accordingly if the acceptor thread count is increased.

Note: The worker-thread count cannot be updated on runtime through the Console as of today. This option will be provided as an option in future.

The worker-thread count can be increased or decreased by editing the `<worker-thread-count>` tag in the `web-config.xml` file:

```
<worker-thread-count max-active="-1" max-keepAlive="150">200</worker-thread-count>
```

The default value for the maximum worker thread count is 200 with **Keep-Alive** count as 150. It is better to have keep-alive count less than the maximum worker thread count. This ensures that if there is a high traffic of requests, requests get served without making some clients starve. Following are the minimum and maximum values that can be provided:

- **Minimum value:** There is no such thing as a minimum value. For a really low request rate with high process time, the count can also be set to a value as low as 1 - if required.

If set to a higher value, the inactive threads consume memory and flood the listener socket to check with new requests. This degrades the VM performance.

Note: The value can be decreased to 1/10th of total number requests received at peak loads collected by statistics.

- **Maximum value:** There is no such thing as a maximum value. For a really high load server, the count can also be set at 1000 or more - if required.

If set to a lower value, a lot of requests will be in queue before they can be processed.

Web sites with many hits but less processing time, static pages, and dynamic pages with lower process times must have high worker thread count.

For Web sites having high processing time with comparably low hit rate, the worker-thread count value must be lower.

Each request requires a single thread. Concurrent processing at high loads requires a higher number of threads. However, the thread count must be kept limited to a lower value depending on the OS and available resources.

Viewing Worker-threads

To view the worker-threads, use the `status` command at the command shell. This displays the number of threads created and the number of threads in waiting.

If compared to the actual number, the number of waiting threads is high, it means a lot of threads are waiting unnecessarily, which may degrade the performance. This is because of extensive context switches, reduced time slices, and memory consumption. Reducing the thread count improves performance.

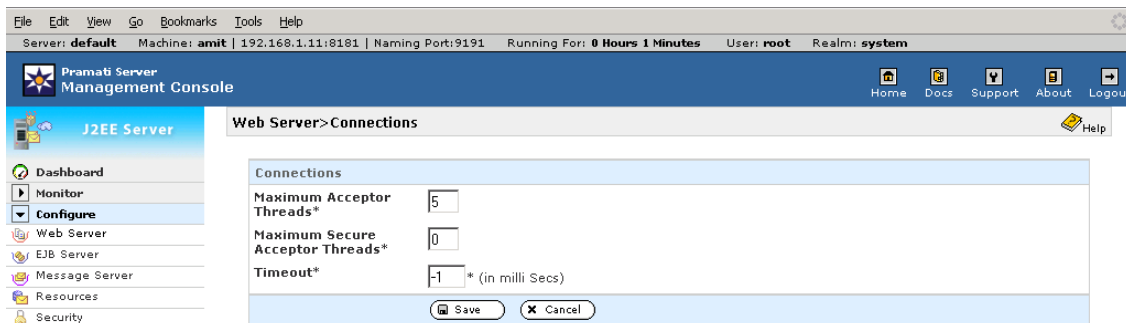
If the number of threads that are waiting are low, some requests are either waiting to get processed or are not getting accepted. Increasing the count improves performance.

Acceptor-thread Count

This value can be modified using the Console. To do this, log on to the Console and select **Configure > Web Server**. The **Connections** section displays the current acceptor thread count.

Fields	Description
Maximum Acceptor Threads	Maximum number of acceptor threads allowed. The default is 5.
Maximum Secure Acceptor Threads	Number of secure acceptor threads.
Timeout	Server socket timeout, specified in milliseconds. The format must be of type positive-int.

To change this value, click the **Edit** icon on the right.



Edit the required values and click **Save**.

Varying Cache Size

Pramati Server provides caching of dynamic content which can be enabled or disabled by editing the `<cache-enabled>` attribute in the following tag in `web-config.xml`:

```
<caching cache-rule-file-name="web-cache.xml" enabled="true"/>
```

The parameters for caching are specified in a separate file as specified in the `cache-rule-file-name` attribute. This is discussed in the next section.

The `cache-size` (in the `web-cache.xml` file) indicates the memory space that is allocated to cache static content.

```
<max-cache-size-mb>10</max-cache-size-mb>
```

The cache size is based on the number of virtual hosts. File I/O operations are generally expensive and are used while serving static content. To avoid loading the same static files, they are cached when served for the first time. When another request for the same file is made while delivering a content from cache, it increases Server performance significantly. A higher cache improves Server throughput when a proper value is set. Setting a very high value reduces the memory available to the VM and degrades performance.

Note: For a file to be cached, the file size must be less than 1/10 of the actual cache size. For a Cache Server, where the client does not make direct requests to the server, it is advised to keep the cache-size low. Server must be provided with maximum possible cache size based on the size of the static content of the application and available RAM.

Dynamic Caching

Pramati Web Server provides rule-based caching for both static and dynamic content for performance enhancements. Dynamic caching can be enabled by the Console. Internally, these changes get affected in the `web-cache.xml` file.

This section looks at how to configure the dynamic caching for applications running on the Server. This file can also be edited manually, but the use of the Console is recommended.

To configure the Web server for Dynamic Caching, log on to the Console and select **Configure > Web Server** in the Explore panel to reach the screen for the configured Hosts. Select **Settings** against the virtual host you want to configure the cache for, and scroll down on the page to the **Cache** section.

Read “Editing Cache Properties for Virtual Hosts” in the Chapter “Configuring Pramati Web Server” of the *Pramati Server Administration Guide* for further details. You can read here the following details:

- Editing Cache Properties
- Adding a Cache Rule
- Configuring Input Parameters
- Configuring Cache Expiry
 - Time-based
 - Custom-based
 - Application-based
- Configuring Cookies
- Configuring Headers

Session Timeout

Timed out sessions show when an idle session is discarded. Sessions are used to maintain a stateful HTTP connection. These sessions contain user profiles and other useful information.

Default Session Timeout

The `<session-config><session-timeout>30</session-timeout></session-config>` tag in the `default-web.xml` specifies the timeout value. The default value is 30 minutes. It can be modified as:

- **Increase to 60 minutes:** Web sites used for online shopping must have a high `session-timeout` value. If lower value is set, these sessions may be discarded prematurely causing inconvenience to users.
- **Decrease to 5-6:** For Web sites providing static information, the session time out may be set to a lower value such as 5 minutes. If set to a higher value, it results in increased memory usage, as the session objects take a long time to be cleaned up.

Viewing Sessions

To view the number of sessions from shell, use the `status` command. This displays the number of sessions created and sessions getting timed out.

For example:

```
----- Session Count
Total Sessions : 10
  Active Sessions : 7
  Inactive Sessions : 30
```

If the number of sessions timed-out are substantially lower than the session count in store, it means that many sessions are active and are consuming a lot of VM memory. In this case, it is better to set the session timeout to a lower value. If the application permits, reduce the session-timeout value. Where higher values are required, it is not possible to reduce it irrespective of performance status. The main aim is to keep the session count low.

Cleaning up Stale Sessions

To improve Server performance, an idle session is discarded.

Session Cleanup Time

The tag `<session-cleanup-time-seconds>` is responsible for cleaning up stale sessions.

```
<session-cleanup-time-seconds>900</session-cleanup-time-seconds>
```

The default value is 900 seconds. For optimal performance, set the value as half of `<session-timeout>` tag.

Network Parameters

The parameters explained in the following section are tuned to enhance the network performance. The network parameters are specified in the `<socket-properties>` tag.

Adjusting Packet Sizes with Traffic

Nagle's algorithm is used to reduce the network traffic by optimizing the packet size being transferred over the network. The values can be modified by editing `web-config.xml` file or by setting values through the Console. Log on to the Console and select **Configure > Web Server** and click **Edit** in the **Socket Properties** section.

Enabling Nagle's Algorithm

`tcp-no-delay` indicates whether or not to enable Nagle's algorithm. Default value is *Yes* and can be modified as:

- Set to yes: For a large sized response such as 10K or more, set the value to *Yes*.
- Set to no: For a smaller sized response of 2K or 3K, set the value as *No* as packet transfer and its confirmation increases the network traffic.

Note: For a large response, use a suitable application buffer management that significantly improves performance over Nagle's algorithm. Refer to RFC 896 (<http://www.faqs.org/rfcs/rfc896.html>) for a detailed discussion on Nagle's algorithm.

Releasing Sockets for Reuse

Socket creations and destruction for each request is one bottleneck while serving Web requests. Using the same socket for subsequent request through persistent connection increases performance. HTTP1.1 protocol mandates support of persistent storage.

idle timeout

The `idle-timeout-millis` tag in `web-config.xml` indicates the time for which a socket is kept waiting for subsequent requests before being closed.

```
<idle-timeout-millis>-1</idle-timeout-millis>
```

The default value is -1 milliseconds (that is, infinite). It can be modified as:

- For clients using the **Connection Keep-Alive** option, `idle-timeout-millis` must be high to ensure that any subsequent request from the same client is processed without creating a new socket.
- When **Keep-Alive** option is set to `false`, or HTTP1.0 protocol is being used, set `idle-timeout-millis` to a lower value.

For a Web site with a number of embedded frames and images referred from main page, pipelining is used. Higher value of the `idle-timeout` parameter improves the performance. For a Web site having more textual information with a few embedded requests, this parameter value must be lower.

Buffer Size for Sending Data

Buffer size indicates the size used to write response data to the client. This option is used by the networking code for the platform to set the underlying network I/O buffers, accordingly. Increasing buffer size can increase the performance of network I/O for high-volume connection, and decreasing the size can reduce backlogging of incoming data.

This value can be changed by editing the `<send-buffer-size-bytes>` tag in the `web-config.xml` or by editing the value using the Console and making it available at runtime.

To edit the value using the Console, select **Configure > Web Server** and click **Edit** in the **Socket Properties** section. The default value is 8192 bytes.

If the size of a response to a client is 10K or more, set this parameter size to 8K to increase the performance of the application.

Buffer Size for Receiving Data

The networking code for the platform uses the buffer size to set the underlying network I/O buffers.

The `<received-buffer-size-bytes>` tag indicates the buffer size used to receive request headers and body information from client. This value can be changed by editing `web-config.xml` or by editing the value using the Console and making it available at runtime.

To edit it using the Console, select **Configure > Web Server** and click **Edit** in the **Socket Properties** section. The default value is 8192 bytes. It can be modified as:

- **Increase:** If the request content length is considerably high, it is advisable to set this to a higher value such as 10K depending upon the request size.
- **Decreased to less than 1K:** For requests with content length less than 1K, the optimum value for the parameter ranges between 0.5-1Kb.

Thread Funneling

Thread funneling is pluggable and can be tuned to boost response time. For more information, read the chapter on *Extreme Threading*.

Tunable parameters for thread funneling are explained in the following sections.

Worker Thread Count

Worker thread count represents the maximum number of requests that can be processed concurrently by the Web container. In `web-config.xml`, the tag appears as

```
<thread-funnel worker_thread_count="50" enabled="false">
```

The default value is 50, minimum value is 1, and any non-negative value can be set as maximum value.

Request Processing

The request processing cycle is divided into processing portions called *activities* depending on the type of resource. Each tunable property represents an activity in the Web request processing cycle. Request processing is a CPU intensive activity. In `web-config.xml`, the tag appears as:

```
<activity-type name="request_processing" num="10"/>
```

The default value is 10, minimum value is 1, and maximum value is the value specified in the `worker_thread_count` tag.

File I/O

The activity File I/O represents the file I/O intensive part of the Web request. In `web-config.xml`, the tag appears as:

```
<activity-type name="file_io" num="30"/>
```

The default value is 30, minimum value is 1, and maximum value is the `worker_thread_count`.

Network

The activity Network represents network I/O intensive part of the Web request. In `web-config.xml`, the tag appears as

```
<activity-type name="network_io" num="30"/>
```

The default value is 30, minimum value is 1, and maximum value is the `worker_thread_count`.

KeepAlive Waits

The activity KeepAlive Waits represents file I/O intensive part of the Web request. In `web-config.xml`, the tag appears as

```
<activity-type name="keepalive_waits" num="40"/>
```

The default value is 40, minimum value is 1, and maximum value is the `worker_thread_count`.

Threshold for Service Unavailable

Thread funneling framework can be tuned to send service unavailable response if the number of waiting threads at the CPU is greater than the value set in the following tag in `web-config.xml`:

```
<service_unavailable_threshold num_waiting="20"/>
```

Priority Processing

Tuning the Server for priority processing and using thread funneling framework, priority users can be served at average response times even under heavy loads. It is not enabled by default.

Configure the following properties in the `web-config.xml`:

- Preferred application name when accessed by priority users

```
<application name="TestpriorityWeb" virtual_host="default">
```
- Realm to which priority must be applied at the specified level

```
<priority-groups realm="system" level="8">
```
- Groups to which priority user can belong

```
<groups>administrator, everybody</groups>
```
- Users that are priority users

```
<users> root, test </users>
</priority-groups>
</application>
```

In the above example, when any user belonging to `groups` with login-name `root`, `test` in the realm `system` accesses application with context `root`, `TestpriorityWeb` is served at priority level 8. The normal priority level is 5.

Another example for configuring priority processing is:

```
<application name="admin" virtual_host="default">
<priority-groups realm="system" level="8">
<!--<groups>administrator, everybody</groups> -->
  <users> root </users>
</priority-groups>
  </application>
</priority-processing>
```

Using the `<acceptor-thread-count>` tag

The *Maximum Acceptor Threads* field is the number of acceptor threads that accept the connections from the clients. This count can be increased or decreased during run-time through the Console, or by editing the `web-config.xml`. The entry in `web-config.xml` is:

```
<acceptor-thread-count>5</acceptor-thread-count>
```

The default value is 5.

Worker Thread Count for the server must be increased accordingly if the acceptor thread count is increased. Worker thread count cannot be updated on runtime through the Console as of today. This option will be provided as an option in future.

Applications make use of resources to serve data from a database. The container maintains a pool of connection objects. Whenever a request arrives for data from the database, the container retrieves an object from the pool. The object executes the queries and retrieves data. The application server keeps connection objects available in a pool, instead of creating the objects on request. This chapter discusses the resource configuration options that are available for tuning the Server.

Most of the datasource parameters can be viewed and modified at runtime through the Console. Each data resource has an associated connection pool with the parameters.

Resource Properties

This section discusses the properties available to tune JDBC resources. These values can be provided at the time of creating the resource and can be modified at runtime using the Console.

Max Pool Size

This represents the maximum number of connection objects that can be present in a pool. Maintaining a pool of datasource objects avoids the overhead of creating an object every time a request for a connection object is made. The default value for the connection pool size is 20.

This value must be set to the maximum number of connections that can be used at any given time. Setting a value lower than this would result in connections waiting for release of other connections that are in use at peak time.

The resource service ensures that the maximum pool size is not exceeded. If the limit is reached, and a new connection request is made when other connections are busy, the connection pool waits till a connection is free.

Min Pool Size

This value represents the minimum number of connections to be retained in the connection pool at any given time. The default value is 5.

For optimum performance, set this value to the average number of connections used at any given time.

Initial Pool Size

This value represents the number of connections the connection pool can have when the datasource is created for the first time. The default value is 1.

The primary purpose of this parameter is to allow faster response times for connection requests when the application uses the datasource pool for the first time.

Idle Timeout

The time interval in seconds after which a connection in the pool is closed if no request arrives. The default value is 300 seconds.

Tuning this value depends on a number of parameters. For example, if connection requests arrive at uneven intervals, this can result in pre-mature close of connections and would require connections to be created on subsequent requests, thus consuming processing time. On the other hand, if this value is set to too high a value, connections would be idle in the pool, consuming memory.

Ideally this value must be set to the average intervals at which connections are requested.

Connection Request Timeout

The time interval in seconds for which an application can wait after making a connection request. After this interval, an exception is thrown to the application. The default value 10 seconds.

This parameter must be configured carefully—setting too low a value might result in application connection requests getting timed out prematurely, while setting too high a value might make a connection request wait for longer than is necessary.

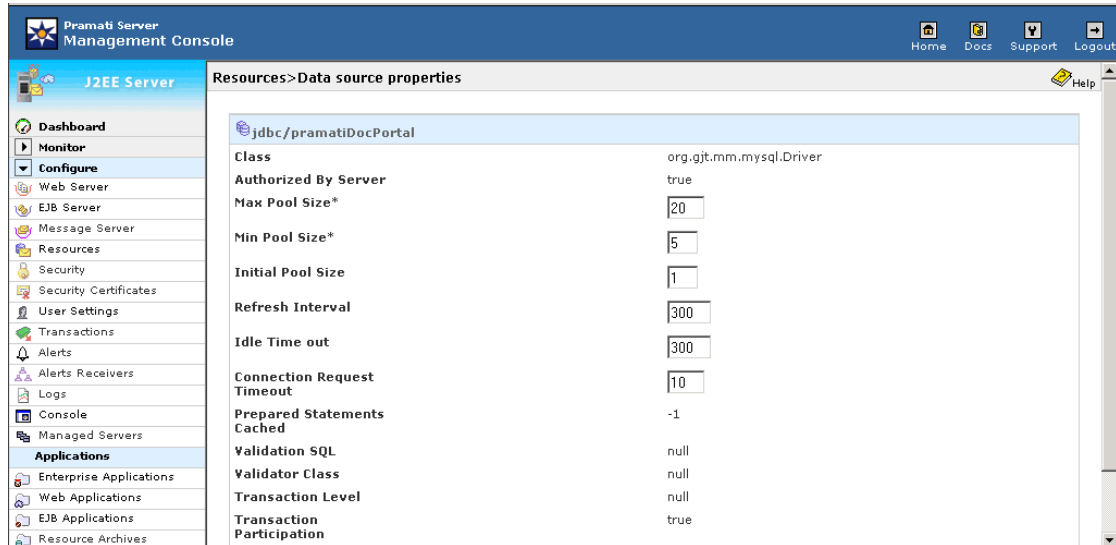
Cached Prepared Statements

This option represents the number of prepared statements that must be cached. Each connection has its own prepared statement cache. Specify here the number of prepared statements to be cached. The value for cache is application dependent. For optimal performance, this value must equal the number of prepared statements in the application. The default value is 20.

In Pramati's implementation, the number of prepared statements in use might exceed the maximum prepared statement cache size, if the number of requests from the application exceeds it. However, on closing the connection the size is reduced to the maximum size set in the `resource-config.xml`.

Tuning Resource Parameters From the Console

As discussed in the previous sections, the Console can be used to modify the resource parameters. To modify these parameters, log on to the Console and select **Configure > Resource**. Click the **Edit** icon in the properties column of the datasource. Modify the parameters according to the requirements of your application. The following screenshot shows this:



In the above page, all parameters except the Prepared Statements Cached property can be edited. To tune this parameter, the `resource-config.xml` file in the `config` directory needs to be edited. However, changes made for this property are not affected at runtime. The Server needs to be restarted.

EJB Container manages resources such as bean pools, caches etc. and provides certain services to effectively serve the EJB requests. Most of the EJB Containers expose tuning parameters that can be fine tuned for efficient working of the Container. These parameters are always set to their optimal default values and the applications may not always want to tweak them. But, the growing needs of an enterprise may require tweaking certain parameters to achieve better application performance.

Pramati EJB Container provides tuning parameters at three different levels or scopes:

- EJB Container Service
- Application
- Enterprise Bean

The EJB Container Tuning parameters can be classified into five categories:

- Container resource pools (Bean pools, thread pools etc.)
- Resource waits
- Time based operations
- Concurrency Strategies
- Persistence options

Container Resource Pools

EJB Container allocates beans, threads and other components while serving an EJB request. As the load on the system increases the usage of these resources also increase and hence the container has to efficiently utilize the system resources.

Tuning Bean Pool Sizes

The Bean pool implemented in Pramati EJB Container comprises of a pool of bean instances and a LRU victimized cache. The configured `max-pool-size` value in the deployment descriptor is shared for both the free pool and the cache.

- Concurrency level on the bean: The size of the bean pool restricts the number of concurrent threads that can operate within the Container. If the pool size is set to a lesser value, the requests queue up at the pool waiting for the bean to get released. It is very essential to set the pool size to an optimal value based on the usage of the beans, which can be determined from the application design.

- **Session Facade:** Session Facade is a well known design pattern adopted by most of the J2EE applications. According to this design pattern, session beans acts as business entry points to the EJB Container. Setting the pool sizes of the facade beans determines the concurrent access to the entire container. The maximum number of concurrent requests served by the container can be effectively determined by the sum of the pool sizes of all facade beans.
- **Resource usage:** The database connection pool sizes must be set based on the bean pool sizes to allow maximum concurrency and avoid deadlocks.

EJB pool is a combination of free pool and a cache and the free pool can be disabled or enabled using the bean level tag `enable-freepool`. Use of free pool improves the performance of bean operations. But, if the application does not clean up the instance fields of an enterprise bean, re-use of instances could lead to reading stale data that has been set in other transactions and in such cases free pool must be disabled.

Max Timer Threads

Resource Waits

Enterprise bean invocation requires certain resources and when the container is unable to provide the required resources, the requests would queue up. To avoid unbounded resource waits, timeouts are specified through configuration parameters. If the requests wait for more than the specified timeout period, EJB container throws an exception indicating a timed out request.

Pool Wait Timeout

Pool timeout indicates the maximum time a client would wait to get a bean instance. A System exception is thrown when a bean is not obtained in the specified time and the current transaction is rolledback.

What do pool timeouts indicate? And how do we handle them?

Pool timeouts occur in the following cases:

- **Configured pool size is low:** Increase the pool size to a sufficiently high value
- **High number of requests to the bean:** If this is the case then the pool timeout exceptions are expected. If there are enough system resources to handle this rush of requests, then increase the pool size.
- **System running slow :** When the system is slow then the transactions lock up the beans for a longer time resulting in pool timeouts.

It is very difficult to identify which of the above have caused a pool timeout. Server monitoring and statistics would give a better idea to identify the above cases.

Primary Key Wait Timeout (Lock Timeout)

Primary Key wait timeout is specified only for Entity beans and indicates the maximum time a bean would wait for a particular Primary Key.

What does lock timeouts indicate? And how do we handle them?

- Configured value of timeout is low: Ideally the value of lock timeout must be same as that of transaction-timeout configured.
- High concurrent requests to the bean: If the number of concurrent requests to a bean are high, then we can expect these timeout exceptions. If the bean is highly concurrent then probably configuring a right concurrency strategy would help solving lock timeouts.
- Application deadlock:
- System running slow: When the system is running slow or if the transactions are taking longer time because of slow down of the the back-end database operations etc.

Time-Based Operations

Session Timeout

Session timeout indicates the maximum time a stateful session bean can exist within the container, available for client requests. When a client makes a request on a stateful session that has timed out, Container throws an exception and discards the session.

With the increase in Internet traffic and complexities of applications, a single server handling multiple requests has become common. When a server is overloaded with requests, it may degrade the efficiency and performance. Load Balancing provides a solution to this problem. The server distributes the load to other machines or nodes that do the task for it.

Pramati Cluster provides a pluggable Load Balancer architecture. Different load balancing algorithms can be used by the cluster. Pramati Server ships with a Weighted Round Robin Load Balancer Algorithm.

This chapter discusses how to tune a clustered scenario for better performance for Web nodes and Java clients.

Load Balancing for Web Nodes

Load Balancing from Web nodes for different EJB nodes in the cluster occurs for every new `InitialContext()` creation. The new `InitialContext (IC)` connects to the next node in the cluster returned by the Load Balancer algorithm.

Load Balancing for Java clients

Load Balancing for every Java client that connects to an EJB node in Pramati Cluster, depends on the Load Balancer Algorithm (LBA). When a Java client connects to a node, all requests are directed to the same EJB node except in case of a failover.

Load Balancer Algorithm

Pramati Server ships with a Weighted Round Robin Load Balancer Algorithm. When configuring a cluster, the default weight is '1'. The weight on each node can be changed while configuring the cluster. For Java clients, a client-based Round Robin algorithm is used.

For example, if Machine A hosting a node is twice as powerful as Machine B hosting another node, A is weighted '2' while B is weighted '1'. If three requests are made, two requests are directed to A and one to B. The LBA calculates the ratios of the weights and connects clients accordingly.

Weighted Round Robin

Special cases of the Weighted Round Robin Algorithm are:

- It is implemented as a Round Robin by default when a cluster is configured. All nodes have weight '1'.
- Setting the weight of a node to '0' makes it a Hot Backup node. This node does not receive any request till all nodes in the cluster with non-zero weights have failed. Hot Backup node stops receiving requests when a weighted node rejoins the cluster.

Setting Load Balancing Properties

Load balancing properties can be set using `com.pramati.naming.distributeload`. The `com.pramati.naming.distributeload` takes the values of:

- VM: When a new IC is created in a VM, the client context attaches itself to one node in the cluster depending on Load Balancer. All subsequent IC are directed to this node.
- IC: Multiple InitialContexts within a VM are directed to different EJB nodes using round robin algorithm.
Note: The Load Balancer that runs in the cluster node is different. The client-based load balancer distributes IC in a round robin fashion.
- LOOKUP: This implies that a lookup based on each lookup of the IC is a sub-set of this option. Apart from alternating nodes for each new IC, each lookup of an EJBHome is rotated.

Customizing Load Balancer

To include the custom Load Balancer, configure the cluster with a fully qualified Load Balancer class name.

For example, `try.sample.lba.SampleLBA`. This class must be in the classpath of the Pramati Server and is added to the `NODE_CLASSPATH` property in the environment properties file located at `<install_dir>/server/nodes/Cluster/<node_name>/config/env.props`. This must be done for every node in the Cluster.

Writing Applications for Load Balancing

The goal of load balancing is to evenly distribute workload between multiple servers. Deploying an application on a cluster from Console is identical to deploying an application on a standalone server. A server synchronizes the deployment across all nodes that form the cluster.

To balance application request loads, place one or more copies of an application component on multiple servers. Each server is configured as a node of the same cluster, allowing it to find application components on other servers.

The Load Balancing and failover mechanisms in a cluster are handled transparently. When deploying an application, it is necessary to decide if the application must be configured for Load Balancing.

To distribute the EJB load among the available EJB nodes, the Web nodes do Load Balancing. This is done by the Naming Service that runs on the Web node. The Naming Service is cluster-aware and keeps track of all EJB nodes in the cluster.

When a new IC is requested, the Naming Service does Load Balancing on the available EJB nodes and attaches the context with the target EJB node. All requests or lookups made on that instance of IC are routed to that node.

If the target EJB node for the context instance fails, a new target node is selected through Load Balancing. The IC instance gets attached to the new node.

To control Load Balancing in JSP pages, Load Balancing can be used. Consider the following piece of code:

```
Context ic = new InitialContext();
Object someBean =
ic.lookup("java:comp/env/ejbs/Mybean1");
Object someOtherBean =
ic.lookup("java:comp/env/ejbs/Mybean2");
```

Here, the EJBs `someBean` and `someOtherBean` are retrieved from the same EJB node. If the code is written as:

```
Context ic = new InitialContext();
Object someBean =
ic.lookup("java:comp/env/ejbs/Mybean1");
ic = new InitialContext();
Object someOtherBean =
ic.lookup("java:comp/env/ejbs/Mybean2");
```

The two beans may not come from the same EJB node. If all the beans come from the same node, the inter-bean calls can use the Fast RMI optimization.

Note: Beans constituting a single session are most likely to participate in a transaction. If the beans are distributed, the transaction too is distributed, which consumes more resources making it expensive.

Distributed transactions must be avoided. It is important that a single session uses a single instance of the IC. For example:

```
HttpSession ssn = request.getSession();
Context ic = (Context)ssn.getValue("mycontext");
if(ic == null){ic = new InitialContext();ssn.putValue("mycontext", ic);}
ic.lookup(.....)
...
```

This code ensures that a single user session is always attached to the same EJB node.

Concurrency Control

The Server manages all complexities of concurrency control in multiple VMs, transparent to the application. There are no additional programming constraints imposed on an application to achieve this control.

Session beans

Session beans are not shared objects and there is no additional requirement to control concurrency in a cluster.

Entity beans

Entity beans use the Remote Lock Server (RLS) to maintain concurrency across nodes. This acts as a central repository for beans to acquire and release locks on Entity beans.

RLS ensures that only one node acquires a lock on an Entity Bean at one time. The Local Lock Service (LLS) communicates with the RLS on behalf of the node and coordinates acquiring and freeing of locks, when instructed by the RLS.

Once a lock on a bean is acquired from RLS, it is not released back to RLS after execution of the business method. The node through the LLS maintains it. This is because a client session is likely to work with a definite set of beans, which may be exclusive with other beans interfacing with other clients. In this case, it is more efficient to maintain the locks locally as the client session attached to this node may further use these beans.

For locks maintained locally, the node does not communicate with the RLS and processes the request faster.

Failover Mechanism

Servers use intelligent wrappers around regular stubs to provide failover capability. These wrappers contain information required for a smooth failover. The wrappers are transparent to the client and require no special programming on the client-side or RMI complier.

Stateless Session Beans

For stateless session beans, the wrapper creates a new instance of the bean on a new EJB node and executes the call on that bean instance.

Stateful Session Beans

For stateful session beans, failover is in-memory or is achieved with the help of a database. This information is taken during cluster configuration. During deployment of the application for stateful session beans, the deployer can specify the methods of the bean that change state. This

information is used by the EJB container to persist the state of the bean at the end of specified methods in-memory or to a database. The database information is taken at the time of configuring the cluster. If the session state is maintained in-memory, the state gets replicated on all the other nodes.

In case of database persistence, all nodes of the cluster use a single database or table to persist the stateful session beans. When an EJB node on which a stateful session bean is created goes down, the wrapper creates a new instance of the bean on a different EJB node. It also attaches the passivated bean instance to the created stateful session bean.

Stateful beans are always persisted after creation. The `ejbCreate()` is treated as a method that changes the state of a bean. Before persisting the state, the container passivates the bean instance making it serializable. After persistence, `ejbActivate()` is called. The bean must ensure that it is in a serializable state after making a call to `ejbPassivate()` and must bring itself back to working state after an `ejbActivate()` call.

Entity Beans

For entity beans, the wrapper recreates a new instance of the bean on a different node through an `ejbFindByPrimaryKey()` method call on the new node. The last committed data is loaded into the bean and the bean is returned to the client for use.

Redeployment Issues

Each deployment of an application on the Pramati Server creates a class loader. When the application is redeployed, a new class loader loads all application classes again and attempts to use the classes from old deployment with the ones from new deployment. This results in `ClassCastException`. This happens when Web clients are connected while redeploying, as these clients may have application object instances stored in their HTTP sessions. All new clients get the classes from the new deployment.

Pramati Naming Service is implemented as a hierarchical tree and is Java Naming and Directory Interface™ (JNDI) compliant. This service provides the applications deployed on Server with:

- Contextual lookup capability
- Caching on client side
- Full support for hierarchy
- Support to bind remote, referencable and serializable objects
- Support to look up UserTransaction object
- Single context factory for clients connecting to standalone Server or cluster

This chapter discusses good practices to use naming service with Server.

Contextual Lookup

Contextual lookup is achieved through an interaction with ‘implicit context manager’. The implicit context manager allows various components of Server to set a context and retrieve it to the executing thread. This context contains:

- Application context
- Module context
- Component context
- Security context
- Transaction context

The EJB and Web containers set the first three contexts that are used by Pramati Naming Service to translate the lookup name. When an application is deployed, the logical names declared by each component are linked to actual JNDI names. In a contextual lookup, the link is resolved and the actual JNDI binding is returned.

Caching on Client-Side

Pramati Naming Service provides caching on client-side. When an object is looked up from the Naming Service, it is cached on client-side and returned for subsequent lookups. Since all Server-side bindings are within the same VM, lookups are treated as local calls and performance is improved. In a cluster environment, caches are intelligently refreshed in case of failure of a Server node.

Support for Hierarchy

Pramati Naming Service supports hierarchy. For example:

A lookup for `ctx.lookup("java:comp/env/jdbc/MyDb");` is split into `Context envCtx = ctx.lookup("java:comp/env");` and `envCtx.lookup("jdbc/MyDb");`.

Also, the lookup `ctx.lookup("myBean/EJB");` may be split into `Context myCtx = ctx.lookup("myBean");` and `myCtx.lookup("EJB");`.

Binding Object Types

Objects that can be bound in the Naming Service are:

- Remote
- Referencable
- Serializable

UserTransaction Support

To lookup the user transaction object in your application, use the standard J2EE name:

```
java:comp/UserTransaction
```

Initial Context Factory

Developers writing clients to Pramati Server must specify an initial context factory `com.pramati.naming.client.PramatiClientContextFactory` in the client code.

This context factory contacts the Server and determines whether the Server is configured as a standalone node or a cluster node and returns the appropriate context to the client. While writing clients, one need not change the context factory depending on Server mode.

The context for standalone node connects to the Naming Service on Server, and all context operations such as lookup, bind, and unbind are performed on it, assuming that all required permissions have been granted.

Normally, only lookup permissions are granted to clients. The context for Pramati cluster contains logic for load balancing and failover.

Properties for InitialContext

When creating an InitialContext from Java or Web clients, various properties can be specified for different behavior of the context:

com.pramati.dns

This specifies the DNS to contact to get the list of server nodes. The DNS configured must be mapped to the ip:port of the EJB nodes. For example,

```
Properties props = new Properties();  
props.put("com.pramati.dns", "www.pramati.com");
```

com.pramati.force.standalone.ctx

Valid values are `true` and `false`. Forces Standalone contexts to be used. This is used when no load balancing and failover logic is required for the client. The default value is `false`.

com.pramati.naming.cacheLookups

Valid values are `true` and `false`. Enables and disables caching of looked up values. The default value is `true`.

com.pramati.naming.distributeLoad

Valid values are `true` and `false`. This is specifically for Pramati Cluster. For Java Clients, this property determines whether load balance is provided for every new `InitialContext()` call. The default value is `false`.

If this property is not set, all new `InitialContext()` calls go to the same EJB node in the Cluster. If set to `true`, every new `InitialContext()` call load-balances to a different EJB node in the Cluster.

The `jndi.properties` file consists of the following parameters:

- `java.naming.factory.initial`
- `java.naming.provider.url`
- `com.pramati.naming.distributeload`

If the Pramati Server name is specified, the parameters can be modified in three ways.

- Directly in the `jndi.properties` file
- System environment: through command line
- Passing each parameter through new `InitialContext()`

com.pramati.naming.cacheLookups

This parameter is common to standalone as well as clustered nodes. Default value is `true`.

The client context caches all looked up objects and improves performance as repeated lookups for the same object become faster due to caching. In a typical production environment, the value must be set to `true` for optimal performance. If the application is redeployed, lookup of a cached object would return a stale copy. It throws an exception from the cache.

In a development environment, where performance is not the objective, this option can be set to `false`. The looked up object returned is always the latest copy.

Before tuning an application server for performance, it is necessary to monitor the resources that may be used by the server and obtain information. Console and Deploy Tool are primarily used for monitoring and tuning. Deployment descriptors and server configuration files may be edited to set the tunable parameters. Information from the server statistics and diagnostic can be used to enhance performance.

This chapter discusses the tools used to monitor activities on the Server. The Console gives a good idea of how much a resource or processing time is being consumed. Third party stress tools can be used to bombard the Server with requests and simulate traffic. The Server can be tuned based on the feedback received.

Server Statistics

Statistics provide data that helps in monitoring Server activities. Pramati Server takes the start and stop time and calculates maximum, minimum, and average values for the activities.

Web container statistics provide request details such as number of requests processed and failed, and help in monitoring Web container activities.

The JMS Server provides statistics for queues, topics, connection lists, and persistence where the maximum, minimum, and average number of messages sent are recorded.

Web Container Statistics

Web container statistics can be categorized as:

- Number of requests: Represents the number of requests made for a particular page in an application
- Sessions created: Number of sessions created for that application
- Response codes: Response codes for the pages in that application
- Cache information: Number of times a cached page was requested, the amount of cached content, their hit ratio etc. Server supports dynamic content caching. Cache rules can be provided for specific applications based on the activity

To view the above information, log on to the Console and select **Monitor > Web Server**.

The screenshot shows the Pramati Server Management Console interface. The left sidebar contains a navigation menu with options like Dashboard, Monitor, Web Server, EJB Server, Message Server, Resources, Security, Transactions, Alerts, Logs, Activity Graphs, Managed Servers, Applications, Enterprise Applications, Web Applications, EJB Applications, Configure, Analyze, and Utilities. The main content area is titled 'Queues' and shows a summary of created and destroyed queues, followed by a table of queue statistics. Below this, there is a section for 'Topics' with a similar summary and table.

Name	Total	Expired messages	Avg queue size	Received(msgs/sec)	Delivered(msgs/sec)
SellQueue	16	0	1	1	1
JMSQueue	0	0	1	1	1
BuyQueue	16	0	1	1	1

Topic	Total Message	Total Subscribers	Expired Messages	Received(msgs/sec)	Avg subscribers
JMSTopic	0	0	0	1	1
StockMarket	32	1	0	1	1

To view the above information using the command shell, use the `status` command. At the Server command prompt, type:

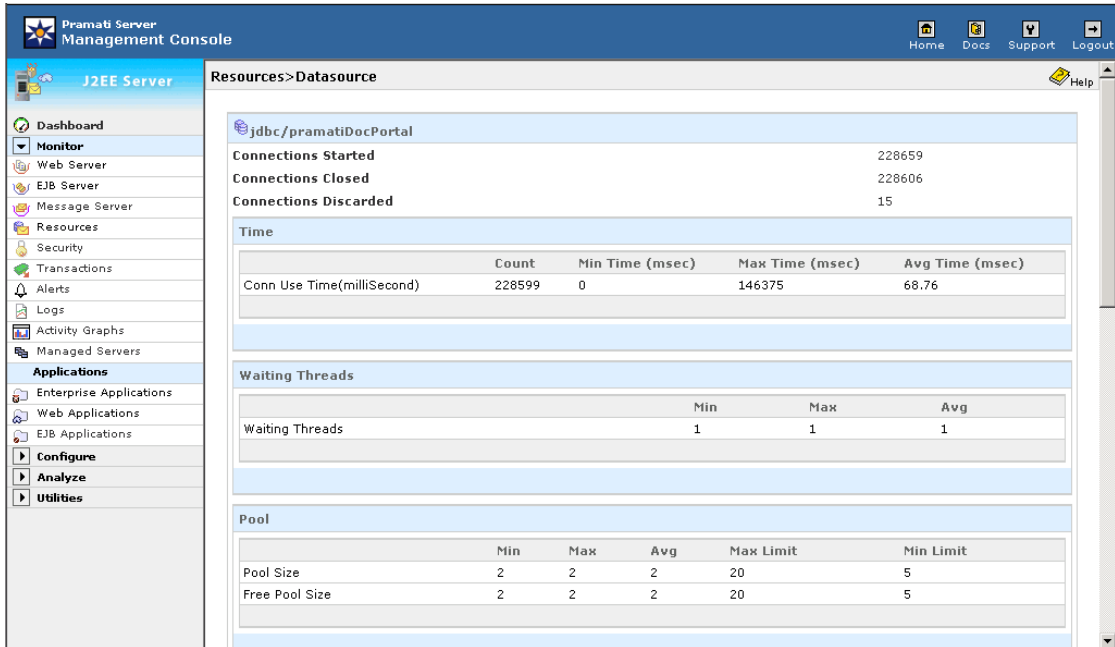
```
j2eadmin@default> status web
```

Resource Statistics

The resource statistics can be categorized as:

- **Connection use time:** Represents the total connection use time, and the minimum, maximum, and average time required to get a connection
- **Waiting threads:** Represents the number of threads waiting on a connection
- **Pool:** Represents the total size of pool including the minimum, maximum, and average number of objects requested. It also informs about the number of objects available in the free pool at a particular time
- **CacheHitRatio:** Prepared Statements improve the performance of a Web application as the statements are cached and do not require re-compiling every time a query needs to be executed
- **Statement execution time:** Number of times a cached prepared statement is executed and the statistics of response as the maximum, minimum, and average time taken to run the statement

To monitor resource statistics, log on to the Console and select **Monitor > Resources** and click the name of the datasource to be monitored.



The screenshot displays the Pramati Server Management Console interface. The left sidebar shows a navigation menu with categories like Dashboard, Monitor, Resources, Security, Transactions, Alerts, Logs, Activity Graphs, Managed Servers, Applications, Configure, Analyze, and Utilities. The main content area is titled "Resources > Datasource" and shows details for the "jdbc/pramatiDocPortal" datasource. It includes summary statistics for connections and a detailed table for connection time.

Time	Count	Min Time (msec)	Max Time (msec)	Avg Time (msec)
Conn Use Time(milliSecond)	228599	0	146375	68.76

Waiting Threads	Min	Max	Avg
Waiting Threads	1	1	1

Pool	Min	Max	Avg	Max Limit	Min Limit
Pool Size	2	2	2	20	5
Free Pool Size	2	2	2	20	5

To monitor Resources using the command shell, type the following command:

```
j2eeadmin@default> status web
```

JMS Statistics

JMS statistics can be categorized as:

- Queue
- Topic
- Connection
- Persistence Store

This information is available from the Console.

The screenshot displays the Pramati Server Management Console interface. The left sidebar contains navigation options such as Dashboard, Monitor, Web Server, EJB Server, Message Server, Resources, Security, Transactions, Alerts, Logs, Activity Graphs, Managed Servers, Applications, Enterprise Applications, Web Applications, EJB Applications, Configure, Analyze, and Utilities. The main content area is divided into two sections: Queues and Topics.

Queues Section:

Name	Total	Expired messages	Avg queue size	Received(msgs/sec)	Delivered(msgs/sec)
SellQueue	16	0	1	1	1
JMSQueue	0	0	1	1	1
BuyQueue	16	0	1	1	1

Topics Section:

Topic	Total Message	Total Subscribers	Expired Messages	Received(msgs/sec)	Avg subscribers
JMSTopic	0	0	0	1	1
StockMarket	32	1	0	1	1

JMS statistics can also be obtained using the `status` command by invoking the JMS shell at the command prompt.

Server Diagnostics

Timing diagnostics in Pramati Server gives an idea about various operations and the performance of Server and application. It enables to identify and analyze an activity, and helps to calculate the time taken by a logical unit of operation. This helps to identify and eliminate bottlenecks at runtime. It provides an inside view of the time taken by various business operations and helps to split time across sub tasks. To know more, read the *Pramati Server Administration Guide*.

Pramati Server provides the following diagnostic data:

- Transaction-based diagnostics: Provides details about total number of transactions committed and rolled-back
- Datasource-based diagnostics: Provides details about the time required by a datasource to obtain connection
- Application-based diagnostics: Provides details about the type of application, time, and date of start of application

Performance Criteria

It is important to ascertain the permissible performance criterion before attempting performance tuning. For Web servers, it is usually the number of requests served each second at a certain load condition. It is crucial to establish the performance criteria in terms of the expected load range on Server and permissible response time.

Permissible response time is difficult to establish as it is qualitative and cannot be determined as a single unique value across all types of requests. Most applications have sections or modules that follow different load characteristics. For example, applications usually contain general public domain content that is static in nature. There might be some dynamic content such as database operations.

Note: More sophisticated testing mechanisms might determine different load conditions for each module. It is important to lay down the requirements against which the exercise would be carried out.

Criteria for Selecting Tools to Test Performance

There are a many products available for load testing a Web server and it is important to select a tool that suits the hardware and OS environment. The load testing tools must support:

- Simulation of any number of clients and users
- Simulating HTTP 1.0 as well as HTTP 1.1 clients
- Simulating user sessions using new cookies or re-using existing cookies
- Simulating authentication when required
- Capturing the response codes and detecting correct responses based on error responses
- Working in a distributed manner

Working in a distributed manner is useful if the load cannot be simulated from a single client machine. In distributed mode, the data and metrics are automatically collected from all agents running on different machines and presented at a central place.

Important: Do not have the Load Simulator and the 'System Under test' running on the same machine. This eats up the system resources. It is important to convert the test to a stand-alone Java client and run it from a separate machine.

Tools and Utilities for Performance Tuning

Web Console

The Web-based Pramati Server Management Console is the primary tool that manages and monitors the Server instances, clusters, and J2EE applications deployed on them.

The Console provides a customizable infrastructure for enterprise server administrators. Dynamic monitoring, tuning, and control of applications enable to optimally deploy resources and extract maximum performance in any deployment scenario.

Deploy Tool

Application parameters such as bean pool size can be set from the Deploy Tool while preparing an application for deployment. jars, wars, and ears can be deployed independently in a target Server environment. Deploy Tool must be aware of the target Server environment properties. To deploy any application, you must first start the Server. The Deploy Tool then connects to the running server and using the Deploy Tool GUI you can deploy the Web or EJB applications.

Server Deployment and Configuration XML Files

Console and Deploy Tool internally setup the Server XMLs. When needed, the XML files can be setup by the administrator.

Server Deployment and Configuration XML Files

- **Deployment XMLs:** The deployment can be done through the XML file. Through `deploy-config.xml` we can configure the deployment of the applicaiton. This can be located at `<install_dir>/server/nodes/<node_name>/config/` directory.
- **Server Configuration XMLs:** Server parameters can be modified by editing `server-config.xml`, located at `<install_dir>/server/nodes/StandAlone/<node_name>/config/`. For clusters, `cluster-config.xml` is located in `<install_dir>/server/nodes/Cluster/<cluster_name>/config/`. Application parameters can be tuned by modifying the:
 - `pramati-j2ee-server.xml` that is application specific and contains parameters for configuring bean pool size.
 - `pramati-or-map.xml` that is entity bean specific and contains O-R mapping information and flags for concurrency and exclusion types.