Oracle® Application Server 10*g*

Performance Guide

10*g* (9.0.4)

Part No. B10379-01

September 2003

Oracle Application Server 10*g* Performance Guide, 10*g* (9.0.4)

Part No. B10379-01

Primary Author: Thomas Van Raalte

Contributors: Eric Belden, Paul Benninghoff, Alice Chan, Greg Cook, Marcelo Goncalves, Helen Grembowicz, Bruce Irvin, Pushkar Kapasi, Paul Lane, Sharon Malek, Valarie Moore, Carol Orange, Julia Pond, Leela Rao, Joan Silverman, Sanjay Singh, Cheryl Smith, Zhunquin Wang, Brian Wright

Contents

Send Us Your Comments xiii

Preface xv

Intended Audience xv
Documentation Accessibility xv
Organization xvi
Related Documentation xvii
Conventions xix

1 Performance Overview

Introduction to Oracle Application Server Performance 1-2
Performance Terms 1-2
What Is Performance Tuning? 1-3
Response Time 1-3
System Throughput 1-5
Wait Time 1-5
Critical Resources 1-5
Effects of Excessive Demand 1-6
Adjustments to Relieve Problems 1-7
Performance Targets 1-7
User Expectations 1-8
Performance Evaluation 1-8
Performance Methodology 1-8
Factors in Improving Performance 1-10

2 Monitoring Oracle Application Server

Overview of Monitoring Oracle Application Server 2-2
Oracle Enterprise Manager Application Server Control 2-2
Oracle Application Server Built-in Performance Metrics 2-2
Native Operating System Performance Commands 2-4
Network Performance Monitoring Tools 2-4
Using Oracle Application Server Built-in Performance Metrics 2-4
Viewing Performance Metrics Using AggreSpy 2-5
Viewing Performance Metrics Using dmstool 2-10
Viewing Performance Metrics Using Spy (for Standalone OC4J) 2-15

3 Monitoring Oracle HTTP Server

Monitoring Oracle HTTP Server with Application Server Control 3-2
Assessing the Oracle HTTP Server Load with Application Server Control 3-2
Investigating Oracle HTTP Server Errors with Application Server Control 3-7
Categorizing Oracle HTTP Server Problems with Application Server Control 3-8
Monitoring Oracle HTTP Server with Built-in Performance Metrics 3-11
Assessing the Oracle HTTP Server Load with Built-in Metrics 3-12
Investigating Oracle HTTP Server Errors with Built-in Metrics 3-16
Categorizing Oracle HTTP Server Performance Problems with Built-in Metrics 3-18

4 Monitoring OC4J

Monitoring OC4J With Application Server Control 4-2
Monitoring OC4J Instances With Application Server Control 4-2
Monitoring J2EE Applications with Application Server Control 4-4
Monitoring OC4J With Built-in Performance Metrics 4-9

5 Optimizing Oracle HTTP Server

TCP Tuning Parameters (for UNIX) 5-2
Tuning Linux 5-4
Setting TCP Parameters 5-6
Network Tuning (for Windows) 5-9
Setting the TCP Protocol 5-10
Setting Server Properties 5-10

Configuring Oracle HTTP Server Directives 5-11
Configuring the MaxClients Directive 5-13
How Persistent Connections Can Reduce httpd Process Availability 5-14
Configuring the ThreadsPerChild Parameter (for Windows) 5-14
Oracle HTTP Server Logging Options 5-15
Access Logging 5-15
Configuring the HostNameLookups Directive 5-15
Error logging 5-16
Oracle HTTP Server Security Performance Considerations 5-16
Oracle HTTP Server Secure Sockets Layer (SSL) Performance Issues 5-16
Oracle HTTP Server Port Tunneling Performance Issues 5-19
Oracle HTTP Server Performance Tips 5-20
Analyze Static Versus Dynamic Requests 5-20
Analyze Time Differences Between Oracle HTTP Server and OC4J Servers 5-20
Beware of a Single Data Point Yielding Misleading Results 5-21
Setting mod_oc4j Load Balancing Policies 5-22
Quick Summary for Using Load Balancing With mod_oc4j 5-23
Using Round Robin and Random Policies With mod_oc4j Load Balancing 5-23
Using Local Affinity Option With mod_oc4j Load Balancing 5-24
Using Weighted Routing Option With mod_oc4j Load Balancing 5-24
Recommendations for Load Balancing With mod_oc4j 5-25

6 Optimizing J2EE Applications In OC4J

OC4J J2EE Application Performance Quickstart 6-2
Improving J2EE Application Performance by Configuring OC4J Instance 6-3
Setting Java Command Line Options (Using JVM and OC4J Performance Options) 6-3
Setting the JVM Heap Size for OC4J Processes 6-4
Setting the JVM Server Option for OC4J Processes 6-6
Setting the JVM AggressiveHeap Option for OC4J Processes 6-6
Setting the JVM Stack Size Option for OC4J Processes 6-7
Setting the JVM Thread Synchronization Option for OC4J Processes 6-7
Setting the JVM Permanent Generation Option for OC4J Processes 6-7
Setting the OC4J DMS Sensors Option 6-8
Setting the OC4J JDBC DMS Statement Metrics Option 6-9
Setting the OC4J Dedicated RMI Context Option 6-10

Setting the OC4J Define Column Type Option 6-10
Using Application Server Control to Change OC4J JVM Command Line Options 6-11
Setting Up Data Sources – Performance Issues 6-13
Emulated and Non-Emulated Data Sources 6-14
Using the EJB Aware Location Specified in Emulated Data Sources 6-14
Setting the Maximum Open Connections in Data Sources 6-15
Setting the Minimum Open Connections in Data Sources 6-16
Setting the Cached Connection Inactivity Timeout in Data Sources 6-17
Setting the Wait for Free Connection Timeout in Data Sources 6-18
Setting the Connection Retry Interval in Data Sources 6-18
Setting the Maximum Number of Connection Attempts in Data Sources 6-19
Setting the JDBC Statement Cache Size in Data Sources 6-19
Setting the JDBC Prefetch Size for a CMP Entity Bean 6-20
Using Application Server Control to Change Data Source Configuration Options 6-20
Setting server.xml Configuration Parameters 6-22
Setting the OC4J Transaction Configuration Timeout in server.xml 6-22
Setting the OC4J Task Manager Granularity in server.xml 6-23
Setting the OC4J Task Manager Check for Updates Control in server.xml 6-24
Setting the OC4J Global Thread Pool Attributes in server.xml 6-24
Setting the OC4J Options for Stateful Session Bean Passivation in server.xml 6-27
Using Application Server Control to Change server.xml Configuration Options 6-28
Improving Servlet Performance in Oracle Application Server 6-28
Improving Performance by Altering Servlet Configuration Parameters 6-28
Servlet Performance Tips 6-30
Improving JSP Performance in Oracle Application Server 6-33
Improving Performance by Altering JSP Configuration Parameters 6-33
Improving Performance by Tuning JSP Code 6-36
Improving EJB Performance in Oracle Application Server 6-40
Configuring Parameters that Apply for All EJBs (Except MDBs) 6-41
Configuring Parameters for CMP Entity Beans 6-43
Configuring Parameters for BMP Entity Beans 6-48
Configuring Parameters for Session Beans 6-50
Configuring Parameters for Message Driven Beans (MDBs) 6-56
Improving Web Services Performance in Oracle Application Server 6-61
Avoiding Web Services Initial Request Delay 6-61

Using Web Services Typed Requests 6-61
Tuning The Web Services Stateful Session Timeout 6-62
Improving BC4J Performance in Oracle Application Server 6-62
Choose the Right Deployment Configuration 6-62
Use Application Module Pooling for Scalability 6-62
Perform Global Framework Component Customization Using Custom Subclasses 6-63
Use SQL-Only and Forward-Only View Objects when Possible 6-63
Do Not Let Your Application Modules Get Too Large 6-64
Use the Right Failover Mode 6-64
Use View Row Spillover to Lower the Memory to Cache a Large Number of Rows 6-65
Choose the Right Style of Bind Parameters 6-65
Implement Query Conditions at Design Time if Possible 6-65
Use the Right JDBC Fetch Size 6-65
Turn off Event Listening in View Objects used in Batch Processes 6-66
Improving JAAS (JAZN) Performance in Oracle Application Server 6-66
Improving JAZN Performance With an XML Provider 6-67
Improving JAZN Performance With an LDAP Provider (Oracle Internet Directory) 6-67
Configuring JAZN Providers 6-68
JAZN Performance Recommendations 6-69
Using Multiple OC4Js, Limiting Connections and Load Balancing 6-69
Configuring Multiple OC4J Processes 6-70
Load Balancing Applications 6-72
Limiting Connections 6-75
Controlling Replication With Multiple OC4Js 6-76
Performance Considerations for Deploying J2EE Applications 6-77
Deployment Performance During the Application Development Phase 6-77
Deployment Performance During the Test and Production Phases 6-79

7 Optimizing OracleAS Web Cache

Use Two CPUs for OracleAS Web Cache 7-2
Configure Enough Memory for OracleAS Web Cache 7-3
Make Sure You Have Sufficient Network Bandwidth 7-7
Set a Reasonable Number of Network Connections 7-7
Connections on UNIX Platforms 7-8
Connections on Windows 7-10

Tune Network-Related Parameters 7-11
Increase Cache Hit Rates 7-13
Check Application Web Server and Web Cache Settings to Optimize Response Time 7-15

8 Optimizing PL/SQL Performance

PL/SQL Performance in Oracle Application Server - Overview 8-2
Process-Based and Thread-Based Operation in Oracle HTTP Server 8-4
Performance Tuning Issues for mod_plsql 8-4
Connection Pooling with mod_plsql 8-5
Closing Pooled Database Sessions 8-7
What Happens to the mod_plsql Connection Pool when the Database Restarts? 8-8
Performance Tuning Areas in mod_plsql 8-8
PL/SQL Based Web Application Development Considerations and Programming Tips. 8-9
Connection Pooling Tips and Oracle HTTP Server Configuration 8-10
Tuning the Number of Database Sessions 8-13
Two-Listener Strategy 8-14
Overhead Problems 8-16
The Flexible Parameter Passing (four-parameter) Overhead 8-18
Using Caching with PL/SQL Based Web Applications 8-18
Using the Validation Technique 8-19
Using the Expires Technique 8-24
System- and User-level Caching with PL/SQL Based Web Applications 8-26
Tuning File System Cache to Improve Caching Performance 8-28
Introducing File System Cache Tuning 8-29
Enabling File System Cache 8-31
Configuring File System Cache to Reside on a Faster File System 8-31
Configuring Session Cache Using Performant File System (OracleAS Portal Only) 8-33
Resizing File System Cache 8-35
Configuring Cache Cleanup 8-38
Oracle HTTP Server Directives 8-39

9 Instrumenting Applications With DMS

Introducing DMS Performance Metrics 9-2
Instrumenting Applications With DMS Metrics 9-2
Monitoring DMS Metrics 9-3

Understanding DMS Terminology (Nouns and Sensors) 9-3
Isolating Expensive Intervals Using PhaseEvent Metrics.... 9-7
Adding DMS Instrumentation To Java Applications 9-8
Including DMS Imports.... 9-9
Organizing Performance Data 9-9
Defining and Using Metrics for Timing 9-10
Defining and Using Metrics for Counting 9-12
Defining and Using Metrics for Recording Status Information.... 9-12
Validating and Testing Applications Using DMS Metrics 9-13
Validating DMS Metrics 9-14
Testing DMS Metrics For Efficiency 9-15
Understanding DMS Security Considerations 9-16
Advanced DMS Topics 9-16
DMS Naming Conventions.... 9-17
DMS Coding Recommendations.... 9-19
Resetting and Destroying Sensors.... 9-20
Dumping DMS Metrics To Files.... 9-20
Conditional Instrumentation Using DMS Sensor Weight 9-20

10 Database Tuning Considerations

Tuning init.ora Database Parameters.... 10-1
Tuning Redo Logs Location and Sizing.... 10-3

A Performance Metrics

Oracle HTTP Server Metrics.... A-2
Oracle HTTP Server Child Server Metrics A-2
Oracle HTTP Server Responses Metrics.... A-3
Oracle HTTP Server Virtual Host Metrics A-3
Aggregate Module Metrics A-4
HTTP Server Module Metrics.... A-4
Oracle HTTP Server mod_oc4j Metrics A-4
JVM Metrics.... A-7
JDBC Metrics A-8
JDBC Driver Metrics.... A-8
JDBC Data Source Metrics.... A-8

JDBC Driver Specific Connection Metrics A-9
JDBC Data Source Specific Connection Metrics A-10
JDBC Driver Statement Metrics A-10
JDBC Data Source Statement Metrics A-11
OC4J Metrics A-12
Web Module Metrics A-12
Web Context Metrics A-13
OC4J Servlet Metrics A-14
OC4J JSP Metrics A-14
OC4J EJB Metrics A-16
OC4J OPMN Info Metrics A-18
OC4J JMS Metrics A-19
JMS Metric Tables A-19
JMS Stats Metric Table A-20
JMS Request Handler Stats A-22
JMS Connection Stats A-22
JMS Session Stats A-23
JMS Message Producer Stats A-24
JMS Message Browser Stats A-24
JMS Message Consumer Stats A-25
JMS Durable Subscription Stats A-25
JMS Destination Stats A-26
JMS Temporary Destination Stats A-26
JMS Store Stats A-27
JMS Persistence Stats A-27
OC4J Task Manager Metrics A-28
mod_plsql Metrics A-29
Portal Metrics A-34
JServ Metrics A-43
Overall JServ Metrics A-43
JServ Zone Metrics A-44
JServ Servlet Metrics A-45
JServ JSP Metrics A-46
Oracle Process Manager and Notification Server Metrics A-48
OPMN_PM Metric Table A-48

OPMN_HOST_STATISTICS Metric Table A-49
OPMN_IAS_INSTANCE Metric Table A-50
OPMN_IAS_COMPONENT Metrics A-50
OPMN ONS Metrics A-52
Oracle Application Server Discoverer Metrics A-53

B Component Performance Links

Oracle Application Server Toplink Performance Information B-2
Oracle Application Server Portal Performance Information B-2
Oracle Application Server ProcessConnect Performance Information B-2
Oracle Application Server Forms Services Performance Information B-2
Oracle Application Server Discoverer Performance Information B-2
Oracle Application Server Wireless Performance Information B-2
Oracle Application Server Reports Services Performance Information B-3

Index

Send Us Your Comments

Oracle Application Server 10*g* Performance Guide, 10*g* (9.0.4)

Part No. B10379-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: appserverdocs_us@oracle.com
- FAX:650-506-7365 Attn: Oracle Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle Application Server 10*g* Performance Guide
  500 Oracle Parkway M/S 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and electronic mail address (optional).

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This guide describes how to monitor and optimize performance, use multiple components for optimal performance, and write highly performant applications in the Oracle Application Server environment.

This preface contains these topics:

- Intended Audience
- Documentation Accessibility
- Organization
- Related Documentation
- Conventions

Intended Audience

Oracle Application Server 10g Performance Guide is intended for Internet application developers, Oracle Application Server administrators, database administrators, and Web masters.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

http://www.oracle.com/accessibility/

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

Chapter 1, "Performance Overview"

This chapter provides an overview of Oracle Application Server performance and tuning concepts.

Chapter 2, "Monitoring Oracle Application Server"

This chapter introduces the available performance monitoring tools, including Oracle Enterprise Manager Application Server Control and the built-in Oracle Application Server performance monitoring tools.

Chapter 3, "Monitoring Oracle HTTP Server"

This chapter discusses monitoring the Oracle HTTP Server using Oracle Enterprise Manager Application Server Control and the built-in performance tools available with Oracle Application Server.

Chapter 4, "Monitoring OC4J"

This chapter discusses monitoring Oracle Application Server Containers for J2EE(OC4J) using Oracle Enterprise Manager Application Server Control and the built-in performance tools available with Oracle Application Server.

Chapter 5, "Optimizing Oracle HTTP Server"

This chapter discusses optimizing HTTP server.

Chapter 6, "Optimizing J2EE Applications In OC4J"

This chapter discusses optimizing J2EE applications running on Oracle Application Server Containers for J2EE.

Chapter 7, "Optimizing OracleAS Web Cache"

This chapter discusses optimizing Web Cache.

Chapter 8, "Optimizing PL/SQL Performance"

This chapter discusses optimizing code using `mod_plsql`.

Chapter 9, "Instrumenting Applications With DMS"

This chapter describes the Oracle Dynamic Monitoring Service (DMS) that enables application developers, support analysts, system administrators, and others to measure application specific performance information. The chapter also shows how to use DMS to instrument Oracle Application Server Java applications.

Appendix A, "Performance Metrics"

This appendix discusses the statistics and metrics used to monitor and analyze the performance of Oracle Application Server components.

Appendix B, "Component Performance Links"

This appendix provides links for performance information on additional Oracle Application Server components.

Related Documentation

For more information, see these Oracle resources:

- *Oracle Application Server 10g Concepts*
- *Oracle Application Server 10g Administrator's Guide*
- *Oracle Application Server 10g Security Guide*
- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server Containers for J2EE User's Guide*
- *Oracle Application Server Web Cache Administrator's Guide*

- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*
- *Oracle Application Server Containers for J2EE Servlet Developer's Guide*
- *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*
- *Oracle9i Database Performance Tuning Guide and Reference*
- *Oracle Application Server 10g PL/SQL Web Toolkit Reference*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://otn.oracle.com/admin/account/membership.html`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://otn.oracle.com/docs/index.htm`

For more information, see these Oracle resources:

- For this release, see information on OracleAS Portal performance at:

  `http://otn.oracle.com/index.html`

  From the Oracle Technology Network main page:

  - Choose the Product link
  - Choose OracleAS Portal under Oracle Application Server

Conventions

This section describes the conventions used in the text and code examples of this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples
- Conventions for Microsoft Windows Operating Systems

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`UPPERCASE monospace (fixed-width) font`	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
`lowercase monospace (fixed-width) font`	Lowercase monospace typeface indicates executables, filenames, directory names, and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user. The `JRepUtil` class implements these methods.
`lowercase italic monospace (fixed-width) font`	Lowercase italic monospace font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (`*`digits`*` [ , `*`precision`*` ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`

Convention	Meaning	Example

`. . .`	Horizontal ellipsis points indicate either:
	■ That we have omitted parts of the code that are not directly related to the example
	■ That you can repeat a portion of the code
		`CREATE TABLE ... AS` *`subquery`*`;`
		
		`SELECT` *`col1`*`,` *`col2`*`, ... ,` *`coln`* `FROM employees;`

. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as shown.	`acctbal NUMBER(11,2);` `acct    CONSTANT NUMBER(4) := 3;`
`Italics`	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/`*`system_password`* `DB_NAME =` *`database_name`*

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

Conventions for Microsoft Windows Operating Systems

The following table describes conventions for Microsoft Windows operating systems and provides examples of their use.

Convention	Meaning	Example
Choose Start >	How to start a program.	To start the Database Configuration Assistant, choose Start > Programs > Oracle - *HOME_NAME* > Configuration and Migration Tools > Database Configuration Assistant.
File and directory names	File and directory names are not case sensitive. The following special characters are not allowed: left angle bracket (<), right angle bracket (>), colon (:), double quotation marks ("), slash (/), pipe (|), and dash (-). The special character backslash (\) is treated as an element separator, even when it appears in quotes. If the file name begins with \\, then Windows assumes it uses the Universal Naming Convention.	
`C:\>`	Represents the Windows command prompt of the current hard disk drive. The escape character in a command prompt is the caret (^). Your prompt reflects the subdirectory in which you are working. Referred to as the *command prompt* in this manual.	`C:\oracle\oradata>`
	The backslash (\) special character is sometimes required as an escape character for the double quotation mark (") special character at the Windows command prompt. Parentheses and the single quotation mark (’) do not require an escape character. Refer to your Windows operating system documentation for more information on escape and special characters.	`C:\>exp scott/tiger TABLES=emp QUERY=\"WHERE job=’SALESMAN’ and sal<1600\"` `C:\>imp SYSTEM/`*`password`*` FROMUSER=scott TABLES=(emp, dept)`
`HOME_NAME`	Represents the Oracle home name. The home name can be up to 16 alphanumeric characters. The only special character allowed in the home name is the underscore.	`C:\> net start Oracle`*`HOME_NAME`*`TNSListener`

Convention	Meaning	Example
ORACLE_HOME and *ORACLE_BASE*	In releases prior to Oracle8i release 8.1.3, when you installed Oracle components, all subdirectories were located under a top level *ORACLE_HOME* directory that by default used one of the following names: ■ C:\orawin95 for Windows 95 ■ C:\orawin98 for Windows 98 This release complies with Optimal Flexible Architecture (OFA) guidelines. All subdirectories are not under a top level *ORACLE_HOME* directory. There is a top level directory called *ORACLE_BASE* that by default is C:\oracle. If you install Oracle9i release 1 (9.0.1) on a computer with no other Oracle software installed, then the default setting for the first Oracle home directory is C:\oracle\ora90. The Oracle home directory is located directly under *ORACLE_BASE*. All directory path examples in this guide follow OFA conventions. Refer to *Oracle9i Database Getting Starting for Windows* for additional information about OFA compliances and for information about installing Oracle products in non-OFA compliant directories.	Go to the *ORACLE_BASE**ORACLE_HOME*\rdbms\admin directory.

1

Performance Overview

This chapter discusses Oracle Application Server performance and tuning concepts.

This chapter contains the following sections:

- Introduction to Oracle Application Server Performance
- What Is Performance Tuning?
- Performance Targets
- Performance Methodology

See Also: *Oracle Application Server 10g Concepts*

Introduction to Oracle Application Server Performance

To maximize Oracle Application Server performance, all components need to be monitored, analyzed, and tuned. In the chapters of this guide, the tools used to monitor performance and the techniques for optimizing the performance of Oracle Application Server components, such as Oracle HTTP Server and Oracle Application Server Containers for J2EE (OC4J) are described.

Performance Terms

Following are performance terms used in this book:

concurrency The ability to handle multiple requests simultaneously. Threads and processes are examples of concurrency mechanisms.

contention Competition for resources.

hash A number generated from a string of text with an algorithm. The hash value is substantially smaller than the text itself. Hash numbers are used for security and for faster access to data.

latency The time that one system component spends waiting for another component in order to complete the entire task. Latency can be defined as wasted time. In networking contexts, latency is defined as the travel time of a packet from source to destination.

response time The time between the submission of a request and the receipt of the response.

scalability The ability of a system to provide **throughput** in proportion to, and limited only by, available hardware resources. A scalable system is one that can handle increasing numbers of requests without adversely affecting response time and **throughput**.

service time The time between the receipt of a request and the completion of the response to the request.

think time The time the user is not engaged in actual use of the processor.

throughput The number of requests processed per unit of time.

wait time The time between the submission of the request and initiation of the request.

What Is Performance Tuning?

Performance must be built in. You must anticipate performance requirements during application analysis and design, and balance the costs and benefits of optimal performance. This section introduces some fundamental concepts:

- Response Time
- System Throughput
- Wait Time
- Critical Resources
- Effects of Excessive Demand
- Adjustments to Relieve Problems

> **See Also:** "Performance Targets" on page 1-7 for a discussion on performance requirements and determining what parts of the system to tune.

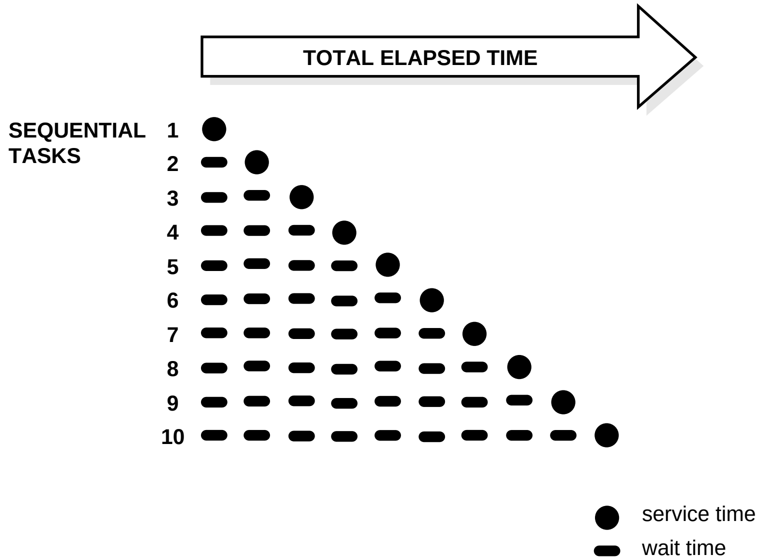
Response Time

Because **response time** equals **service time** plus **wait time**, you can increase performance in this area by:

- Reducing **wait time**
- Reducing **service time**

Figure 1–1 illustrates ten independent sequential tasks competing for a single resource as time elapses.

Figure 1–1 Sequential Processing of Independent Tasks



In the example shown in Figure 1–1, only task 1 runs without waiting. Task 2 must wait until task 1 has completed; task 3 must wait until tasks 1 and 2 have completed, and so on. Although the figure shows the independent tasks as the same size, the size of the tasks will vary.

In parallel processing with multiple resources, more resources are available to the tasks. Each independent task executes immediately using its own resource and no **wait time** is involved.

The Oracle HTTP Server processes requests in this fashion, allocating client requests to available `httpd` processes. The `MaxClients` directive specifies the maximum number of `httpd` processes simultaneously available to handle client requests. When the number of processes in use reaches the `MaxClients` value, the server refuses connections until requests are completed and processes are freed.

> **See Also:** Chapter 5, "Optimizing Oracle HTTP Server"

System Throughput

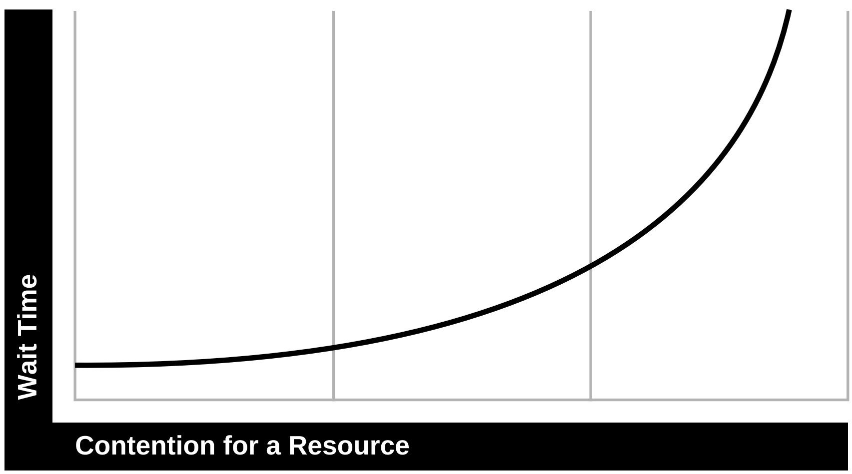
System **throughput** is the amount of work accomplished in a given amount of time. You can increase **throughput** by:

- Reducing **service time**
- Reducing overall **response time** by increasing the amount of scarce resources available. For example, if the system is CPU bound, then adding CPU resources should improve performance.

Wait Time

While the **service time** for a task may stay the same, **wait time** will lengthen with increased **contention**. If many users are waiting for a service that takes one second, the tenth user must wait 9 seconds. Figure 1–2 shows the relationship between **wait time** and resource **contention**. In the figure, the graph illustrates that wait time increases exponentially as contention for a resource increases.

Figure 1–2 Wait Time Rising With Increased Contention for a Resource



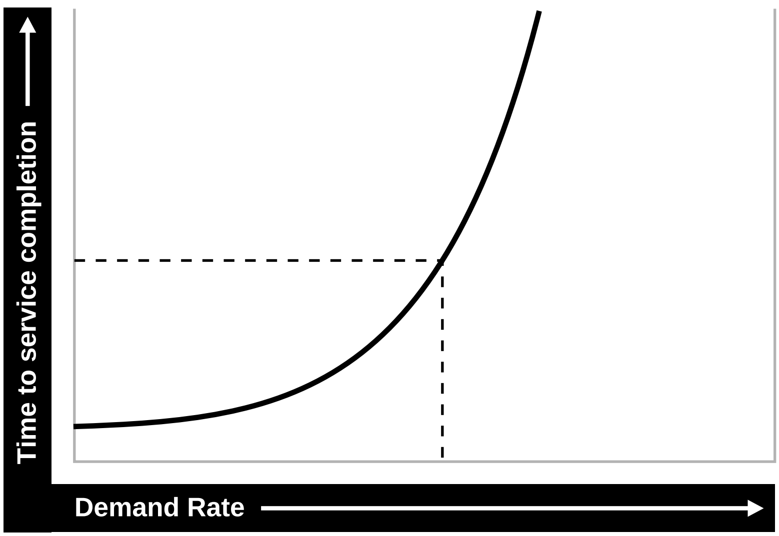
Critical Resources

Resources such as CPU, memory, I/O capacity, and network bandwidth are key to reducing **service time**. Adding resources increases **throughput** and reduces **response time**. Performance depends on these factors:

- How many resources are available?
- How many clients need the resource?
- How long must they wait for the resource?
- How long do they hold the resource?

Figure 1–3 shows the relationship between time to service completion and demand rate. The graph in the figure illustrates that as the number of units requested rises, the time to service completion increases.

Figure 1–3 Time to Service Completion Versus Demand Rate



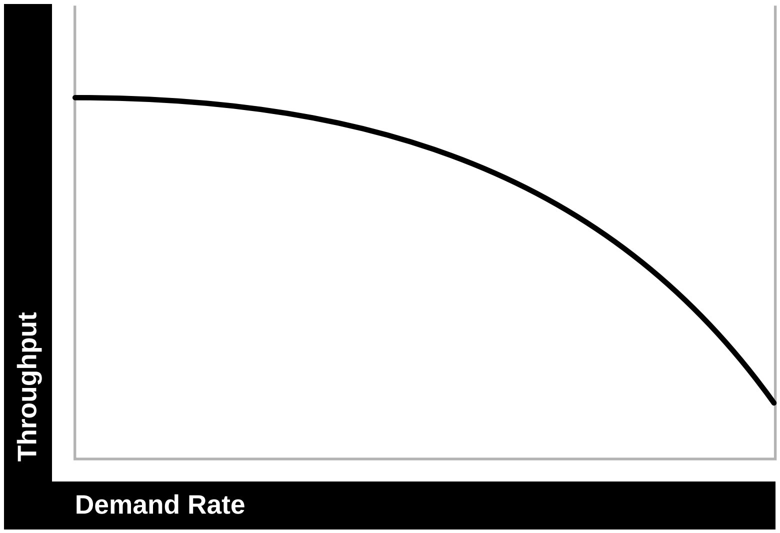
To manage this situation, you have two options:

- Limit demand rate to maintain acceptable response times
- Add resources

Effects of Excessive Demand

Excessive demand increases **response time** and reduces **throughput**, as illustrated by the graph in Figure 1–4.

Figure 1–4 Increased Demand/Reduced Throughput



If the demand rate exceeds the achievable **throughput**, then determine through monitoring which resource is exhausted and increase the resource, if possible.

Adjustments to Relieve Problems

Performance problems can be relieved by making adjustments in the following:

- unit consumption

  Reducing the resource (CPU, memory) consumption of each request can improve performance. This might be achieved by pooling and caching.

- functional demand

  Rescheduling or redistributing the work will relieve some problems.

- capacity

  Increasing or reallocating resources (such as CPUs) relieves some problems.

Performance Targets

Whether you are designing or maintaining a system, you should set specific performance goals so that you know how and what to optimize. If you alter parameters without a specific goal in mind, you can waste time tuning your system without significant gain.

An example of a specific performance goal is an order entry **response time** under three seconds. If the application does not meet that goal, identify the cause (for example, I/O **contention**), and take corrective action. During development, test the application to determine if it meets the designed performance goals.

Tuning usually involves a series of trade-offs. After you have determined the bottlenecks, you may have to modify performance in some other areas to achieve the desired results. For example, if I/O is a problem, you may need to purchase more memory or more disks. If a purchase is not possible, you may have to limit the **concurrency** of the system to achieve the desired performance. However, if you have clearly defined goals for performance, the decision on what to trade for higher performance is easier because you have identified the most important areas.

User Expectations

Application developers, database administrators, and system administrators must be careful to set appropriate performance expectations for users. When the system carries out a particularly complicated operation, **response time** may be slower than when it is performing a simple operation. Users should be made aware of which operations might take longer.

Performance Evaluation

With clearly defined performance goals, you can readily determine when performance tuning has been successful. Success depends on the functional objectives you have established with the user community, your ability to measure whether or not the criteria are being met, and your ability to take corrective action to overcome any exceptions.

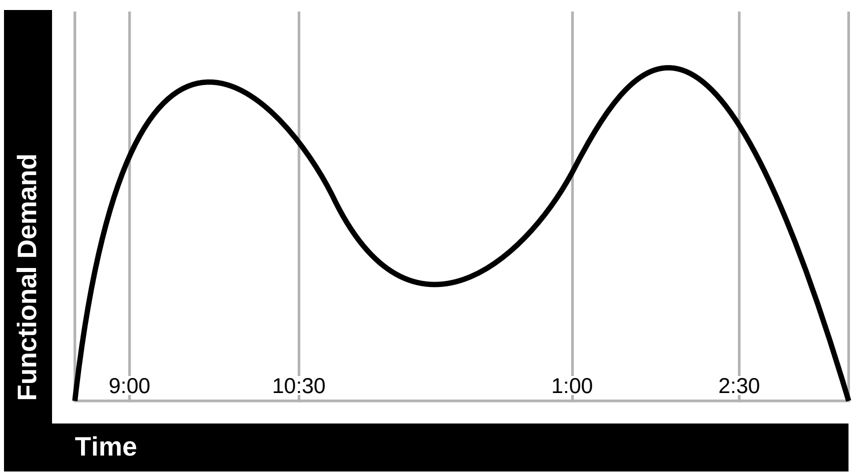
Ongoing performance monitoring enables you to maintain a well tuned system. Keeping a history of the application's performance over time enables you to make useful comparisons. With data about actual resource consumption for a range of loads, you can conduct objective **scalability** studies and from these predict the resource requirements for anticipated load volumes.

Performance Methodology

Achieving optimal effectiveness in your system requires planning, monitoring, and periodic adjustment. The first step in performance tuning is to determine the goals you need to achieve and to design effective usage of available technology into your applications. After implementing your system, it is necessary to periodically monitor and adjust your system. For example, you might want to ensure that 90% of the users experience **response times** no greater than 5 seconds and the maximum **response time** for all users is 20 seconds. Usually, it's not that simple. Your application may include a variety of operations with differing characteristics and acceptable response times. You need to set measurable goals for each of these.

You also need to determine variances in the load. For example, users might access the system heavily between 9:00am and 10:00am and then again between 1:00pm and 2:00pm, as illustrated by the graph in Figure 1–5. If your peak load occurs on a regular basis, for example, daily or weekly, the conventional wisdom is to configure and tune systems to meet your peak load requirements. The lucky users who access the application in off-time will experience better **response times** than your peak-time users. If your peak load is infrequent, you may be willing to tolerate higher **response times** at peak loads for the cost savings of smaller hardware configurations.

Figure 1–5 Adjusting Capacity and Functional Demand



Factors in Improving Performance

Performance spans several areas:

- Sizing and configuration: Determining the type of hardware needed to support your performance goals.
- Parameter tuning: Setting configurable parameters to achieve the best performance for your application.
- Performance monitoring: Determining what hardware resources are being used by your application and what **response time** your users are experiencing.
- Troubleshooting: Diagnosing why an application is using excessive hardware resources, or why the **response time** exceeds the desired limit.

2

Monitoring Oracle Application Server

This chapter discusses how to monitor the performance of Oracle Application Server and its components. Monitoring Oracle Application Server and obtaining performance data can assist you in tuning the system and debugging applications with performance problems.

This chapter contains the following topics:

- Overview of Monitoring Oracle Application Server
- Using Oracle Application Server Built-in Performance Metrics

Overview of Monitoring Oracle Application Server

This section describes how to use the Oracle Application Server tools for performance monitoring. You can monitor the server and its components using one or more of the following:

- Oracle Enterprise Manager Application Server Control
- Oracle Application Server Built-in Performance Metrics
- Native Operating System Performance Commands
- Network Performance Monitoring Tools

Oracle Enterprise Manager Application Server Control

Oracle Enterprise Manager Application Server Control (Application Server Control) allows you to monitor Oracle Application Server and its components. Application Server Control shows performance metrics for Oracle Application Server components, including:

- Oracle HTTP Server (OHS)
- Oracle Application Server Containers for J2EE (OC4J) and Applications running under OC4J
- Oracle Application Server Web Cache
- Oracle Application Server Portal (OracleAS Portal)

See Also:

- Chapter 3, "Monitoring Oracle HTTP Server"
- Chapter 4, "Monitoring OC4J"
- *Oracle Application Server 10g Administrator's Guide*
- *Oracle Application Server Portal Configuration Guide*

Oracle Application Server Built-in Performance Metrics

Oracle Application Server automatically measures runtime performance and collects metrics for the Oracle HTTP Server, including child servers, and Oracle Application Server Containers for J2EE (OC4J) servers. The server performance metrics are measured automatically and continuously using performance instrumentation inserted into the implementations of Oracle Application Server components. The performance metrics are automatically enabled; you do not need to set options or perform any extra configuration to collect them (for performance reasons the JDBC metrics are enabled by setting options).

The Oracle HTTP Server performance metrics enable you to do the following:

- Monitor the duration of important phases of Oracle HTTP Server request processing.
- Collect status information on Oracle HTTP Server requests. For example, you can monitor the number of requests being handled at any given moment.

The OC4J performance metrics allow you to monitor the performance of J2EE containers and enable you to do the following:

- Monitor the number of active servlets, JSPs, EJBs, and EJB methods.
- Monitor the time spent processing an individual servlet, JSP, EJB, or EJB method.
- Monitor the sessions and JDBC connections associated with servlets, JSPs, EJBs, or EJB methods.
- Monitor OC4J JMS events and status.

You can use the performance metrics while troubleshooting Oracle Application Server components to help locate bottlenecks, identify resource availability issues, or help tune your components to improve throughput and response times.

Note: You can use the commands that access the built-in metrics in scripts or in combination with other monitoring tools to gather performance data or to check application performance.

See Also:

- "Using Oracle Application Server Built-in Performance Metrics" on page 2-4
- Appendix A, "Performance Metrics"

Native Operating System Performance Commands

In order to solve performance problems or to monitor your system's activity, you can use the available native operating system commands. Native operating system commands allow you to gather and monitor CPU utilization, paging activity, swapping, and other system activity information.

> **See Also:** Refer to the system level documentation for information on native operating system monitoring commands.

Network Performance Monitoring Tools

You can use network monitoring tools to verify the status of requests that access your Oracle Application Server components. Tools are available that allow you to examine and save network traffic information. These tools can be helpful in analyzing and solving performance problems.

Using Oracle Application Server Built-in Performance Metrics

You can monitor performance using Application Server Control or by viewing the Oracle Application Server built-in performance metrics.

This section describes how to view the built-in performance metrics using the `AggreSpy` servlet, the `Spy` servlet, or using the `dmstool` command.

This section covers the following:

- Viewing Performance Metrics Using AggreSpy
- Viewing Performance Metrics Using dmstool
- Viewing Performance Metrics Using Spy (for Standalone OC4J)

Table 2–1 summarizes the tools that allow you to view built-in performance metrics.

Table 2–1 Oracle Application Server Built-in Monitoring Commands

Command	Description
`AggreSpy`	`AggreSpy` is a pre-packaged servlet that reports performance metrics for an Oracle Application Server instance. You can only run `AggreSpy` when an OC4J instance is configured to support it. By default the OC4J instance named home supports `AggreSpy`.
`dmstool`	Allows you to monitor a specific performance metric, a set of performance metrics, or all performance metrics. Options allow you to specify a reporting interval to report the requested metrics. This command also allows you to show a text report listing all the built-in performance metrics available on the site. `dmstool` is located in the `$ORACLE_HOME/bin` directory.
`Spy`	Spy is a pre-packaged servlet that reports performance metrics for OC4J standalone. `Spy` reports performance data for an OC4J process.

See Also: Appendix A, "Performance Metrics"

Viewing Performance Metrics Using AggreSpy

The `AggreSpy` Servlet displays metrics for Oracle Application Server processes, including Oracle HTTP Server, OC4J, Oracle Process Manager and Notification Server, and other Oracle Application Server component processes.

This section covers the following topics:

- Using the AggreSpy Display
- AggreSpy URL With a Proxy Server
- AggreSpy URL and Access Control
- AggreSpy Limitation When Using Load Balancing With Clusters

Using the AggreSpy Display

`AggreSpy` organizes metrics into two areas: DMS Spies and Metric Tables.

- DMS Spies show the available metrics by parent process type and parent process number. By selecting individual DMS Spies, you can view, in text form, all metrics collected for the associated process.
- Metric Tables show the available metrics by metric table type and when multiple OC4Js are running include OC4J metrics from multiple OC4J instances. By selecting individual metric tables you can view, in table form, all metrics of a specified type. For example, metric tables allow you to view the metrics associated with OC4J Servlets, Oracle HTTP Server Modules, and Oracle Process Manager and Notification Server processes.

Note: To view DMS metrics using `AggreSpy`, you may need to configure your browser to disable the use of a proxy for the localhost, if your system is configured to use proxies. By default Oracle Application Server only allows access for the localhost. See "AggreSpy URL With a Proxy Server" on page 2-7 for details.

DMS metric tables are identified by a name, such as `ohs_server` for the Oracle HTTP Server metrics. In `AggreSpy`, the term Metric Tables refers to the built-in performance metric table names.

You can access performance metrics using `AggreSpy` from the following URL:

```
http://host:port/dms0/AggreSpy
```

where:

host is the Oracle HTTP Server host, for example, `tv.us.oracle.com`.

port is the Oracle HTTP Server listener port, for example `7778`.

Note: You can only run `AggreSpy` when an OC4J instance is configured to support it, and the instance is running. By default, the OC4J instance named home supports `AggreSpy`.

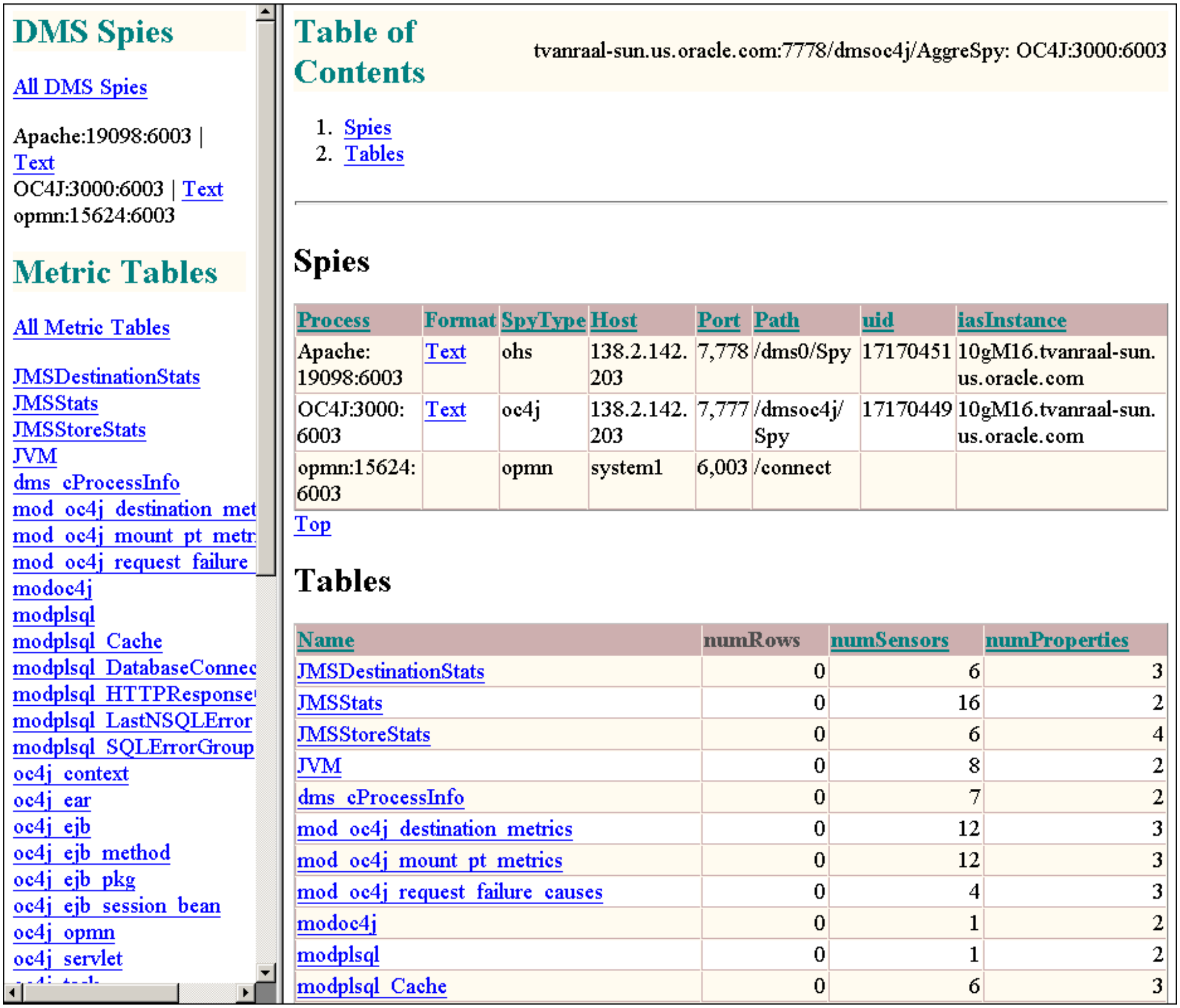
Figure 2–1 shows a sample `AggreSpy` display. The display shows two frames, one containing a list of DMS Spies and DMS Metric Tables, and one showing selected values for the DMS Spies or the Metric Tables.

`AggreSpy` provides navigation and display options, including:

- Access DMS Spies and Metric Tables using the links in the left frame.
- Sort rows in metric tables by clicking on the column headings.
- Display a table containing the descriptions of a Metric Table's metrics by clicking the Metric Definitions link shown on each metric table.

You need to refresh your browser to display built-in metric data after you start `AggreSpy`. When you first use `AggreSpy` many of the fields, and the complete list of DMS Spies and may not contain all of the current Metric Tables. If you wait a short time, and then refresh the display, the data is available and `AggreSpy` shows the complete list of Metric Tables.

Figure 2–1 AggreSpy Performance Metric Display



AggreSpy URL With a Proxy Server

If your browser is configured to use a proxy server, then to access `AggreSpy` on the localhost, you need to configure the browser to disable the use of proxies for the localhost. The exact steps required to disable the use of a proxy server for the localhost depends on the browser you use.

AggreSpy URL and Access Control

By default, the `dms0/AggreSpy` URL is redirected and the redirect location is protected, allowing only the localhost (127.0.0.1) to access the `AggreSpy` Servlet.

To view metrics from a system other than the localhost you need to change the DMS configuration for the system that is running the Oracle Application Server that you want to monitor by modifying the file `$ORACLE_HOME/Apache/Apache/conf/dms.conf` on UNIX, or `%ORACLE_HOME\Apache\Apache\conf\dms.conf` on Windows systems. Example 2–1 shows a sample default configuration from `dms.conf`. This configuration limits `AggreSpy` to only access metrics on the localhost (127.0.0.1) . The port shown, 7200, may differ on your installation.

Example 2–1 dms.conf Sample for localhost Access for DMS Metrics

```
# proxy to DMS AggreSpy
Redirect /dms0/AggreSpy http://localhost:7200/dmsoc4j/AggreSpy
#DMS VirtualHost for access and logging control
Listen 127.0.0.1:7200
OpmnHostPort http://localhost:7200
<VirtualHost 127.0.0.1:7200>
ServerName 127.0.0.1
```

By changing the `dms.conf` configuration to specify the host that provides, or serves DMS metrics, you can allow users on systems other than the localhost to access the DMS metrics from the location http://*host*:*port*/dms0/AggreSpy.

Example 2–2 shows a sample updated `dms.conf` that allows access from a system other than the localhost (127.0.0.1) .

Note: Modifying `dms.conf` has security implications. Only modify this file if you understand the security implications for your site. By exposing metrics to systems other than the localhost, you allow other sites to potentially view critical Oracle Application Server internal status and runtime information.

To view metrics from a system other than the localhost (127.0.0.1) , do the following:

1. Modify `dms.conf` by changing the entries with the value "localhost" or "127.0.0.1" shown in Example 2–1 to the name of the server providing the metrics (obtain the server name from the `ServerName` directive in the `httpd.conf` file, for example `tv.us.oracle.com`).

2. Restart, or stop and start the Oracle HTTP Server using Application Server Control or using the Oracle Process Manager and Notification Server `opmnctl` command. For example,

   ```
   %opmnctl restartproc process-type=HTTP_Server
   ```

   or

   ```
   %opmnctl stopproc process-type=HTTP_Server
   %opmnctl startproc process-type=HTTP_Server
   ```

Example 2–2 dms.conf Sample for Remote Host Access for DMS Metrics

```
# proxy to DMS AggreSpy
Redirect /dms0/AggreSpy http://tv.us.oracle.com:7200/dmsoc4j/AggreSpy
#DMS VirtualHost for access and logging control
Listen tv.us.oracle.com:7200
OpmnHostPort http://tv.us.oracle.com:7200
<VirtualHost tv.us.oracle.com:7200>
ServerName tv.us.oracle.com
```

> **See Also:** *Oracle Application Server 10g Security Guide* for information on Oracle HTTP Server access control

AggreSpy Limitation When Using Load Balancing With Clusters

`AggreSpy` does not work as expected when using Oracle Application Server Clusters. When using a cluster, the Oracle HTTP Server `mod_oc4j` component load balances OC4J requests across Oracle Application Server instances. In this case, `AggreSpy` may report results for systems that are not the localhost (127.0.0.1) .

> **Note:** It is recommended, when using Oracle Application Server Clusters, that you use `dmstool` instead of `AggreSpy`.

Viewing Performance Metrics Using dmstool

The `dmstool` command allows you to view a specific performance metric, a set of performance metrics, or all performance metrics for an Oracle Application Server instance. The `dmstool` command also supports an option that allows you to set a reporting interval, specified in seconds, to report updated metrics every *t* seconds.

For example, you can monitor the performance of a specific servlet, JSP, EJB, EJB method, or database connection and you can request periodic snapshots of metrics specific to these components.

The format for using `dmstool` to display built-in performance metrics is:

```
% dmstool [options] metric metric ...
```

or

```
% dmstool [options] -list
```

or

```
% dmstool [options] -dump
```

Table 2–2 lists the `dmstool` command-line *`options`*. Following Table 2–2 this section presents examples that show sample usage with specific performance metrics. The `dmstool` command is located in the `$ORACLE_HOME/bin` directory on UNIX or in `%ORACLE_HOME%\bin` directory on Windows.

Note: You can use `dmstool` in scripts or in combination with other monitoring tools to gather performance data, to check application performance, or to build tools that modify your system based on the values of performance metrics.

See Also:

"Using dmstool to List the Names of All Metrics" on page 2-12

Appendix A, "Performance Metrics" for a list and description of the DMS metrics

Access Control for dmstool

By default, `dmstool` shows metrics only when it is run from the localhost (127.0.0.1) . If you want to view metrics from an Oracle Application Server running on a remote host, then you need to use `dmstool` with the `-a` option, on the local host, and update the dms.conf file of the remote Oracle Application Server instance in the $ORACLE_HOME/Apache/Apache/conf/ directory on UNIX or %ORACLE_HOME%\Apache\Apache\conf\ directory on Windows.

The configuration changes required to control the access to metrics using dmstool are the same as those for accessing dms0/AggreSpy.

See Also: "AggreSpy URL and Access Control" on page 2-8

Table 2–2 dmstool Command-line Options

Option	Description

-a[ddress] opmn://*host*[:*port*]	By default, without the -a option, dmstool gets metrics from the Oracle Application Server instance with the same $ORACLE_HOME as dmstool. When dmstool runs in the same $ORACLE_HOME as the Oracle Process Manager and Notification Server, OPMN, the –a option is not required.
	
	You can specify –a with the opmn:// prefix and the arguments shown to monitor the Oracle Application Server processes under OPMN control that produce DMS metrics (some OPMN controlled processes, for example Oracle Application Server Web Cache, do not expose DMS metrics).
	
	Where:
	
	host is the domain name or IP address of the host on which the OPMN process is running.
	
	port specifies the OPMN request port that supplies metrics. The request port is specified in $ORACLE_HOME/opmn/conf/opmn.xml.
	
	For example, the following shows the specification in opmn.xml for a request port (request="6003"):
	
	`<notification-server>`
	`<port local="6100" remote="6200" request="6003"/>`
	`.`
	`.`
	`</notification-server>`
	
	Note, if you use dmstool -a to request metrics from a remote system, the system must be configured to provide metrics (by default you can access DMS metrics on the localhost).
	
	See Also: "AggreSpy URL and Access Control" on page 2-8

-c[ount] *num*	Specifies the number of times to retrieve values when monitoring metrics. If not specified, dmstool continues retrieving metric values until the process is stopped. The –count option is not used with the –list option.

Table 2–2 (Cont.) dmstool Command-line Options

Option	Description
`–dump [format=xml]`	Using `dmstool` with the `-dump` option reports all the available metrics on the standard output. Oracle recommends that you run with the `-dump` option periodically, such as every 15 to 20 minutes, to capture and save a record of performance data for your Oracle Application Server server. The -dump option also supports the `format=xml` query. Using this query at the end of the command line supplies the metric output in XML format.
`–help`	List the `dmstool` command-line options.
`–i[nterval]` *`secs`*	Specifies the number of seconds to wait between metric retrievals. The default is 5 seconds. The `interval` argument is not used with the `–list` option. The interval specified is approximate. Note: if the system load is high, the actual interval may vary from the interval specified using the `–interval` option.
`–l[ist] [-table]`	Generates a list of all metrics available. Use the `–list` option with the `–table` option to display a list of all the metric table names. Note, including metric names on the command-line is not valid when using the `–list` option with `dmstool`.
`–table` *`metric_table`*	Includes all the performance metrics for the specified metric table with the name, *`metric_table`*. See Appendix A, "Performance Metrics" or run `AggreSpy` for a list of metric table names.

Using dmstool to List the Names of All Metrics

Every Oracle Application Server performance metric has a unique name. Using `dmstool` with the `–list` option produces a list of all metric names. The `–list` output contains the metric names that you can use with `dmstool` to request monitoring information for a specific metric or set of metrics.

Using the following command, `dmstool` displays a list of all metrics available on the server:

```
% dmstool -list
```

This command displays a list of the available metrics.

See Also: Appendix A, "Performance Metrics"

Using dmstool to Report Values for Specific Performance Metrics

To monitor a specific metric or set of metrics, use `dmstool` and include the metric name on the command-line. For example, to monitor the time the JVM has been running, perform the following steps:

1. Use `dmstool` with the `-list` option to find the name of the metric that shows the JVM uptime:

   ```
   % dmstool -list | grep JVM/upTime.value
   /system1/OC4J:3000:6004/JVM/upTime.value
   ```

2. Use `dmstool` and supply the full metric name as an argument to show the metric value:

   ```
   % dmstool /system1/OC4J:3000:6004/JVM/upTime.value
   Tue Apr 29 16:20:05 PDT 2003
   /system1/OC4J:3000:6004/JVM/upTime.value     14022008        msecs
   ```

Using `dmstool`, the default repeat interval is 10 seconds, so this command shows the updated metric value every 5 seconds. Use the `-count` option to limit the number of times `dmstool` reports values. For example:

```
% dmstool /system1/OC4J:3000:6004/JVM/upTime.value -count 2
Tue Apr 29 16:20:05 PDT 2003
/system1/OC4J:3000:6004/JVM/upTime.value     14336273        msecs

Tue Apr 29 16:20:15 PDT 2003
/system1/OC4J:3000:6004/JVM/upTime.value     14345881        msecs
```

Using dmstool With the Interval and Count Options

To monitor the load balance between the two identified OC4J processes for two hours, use the following `dmstool` command, supplying several metric names on the command-line:

```
% dmstool -i 60 -c 120 \
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed \
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojsp/JSPs/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojsp/processRequest.completed \
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/processRequest.completed
```

This command reports 120 sets of output for the metrics listed on the command line, while collecting data at intervals of 60 seconds:

```
Mon Nov 19 17:13:01 PDT 2002
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed          437 ops
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed                  441 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed          432 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed                  436 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/JSPs/processRequest.completed  452 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/processRequest.completed      425 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/processRequest.completed                455 ops

Mon Nov 19 17:14:01 PDT 2002
/myhost/OC4J:3000:6003/oc4j/default/WEBs/default/processRequest.completed          452 ops
/myhost/OC4J:3000:6003/oc4j/default/WEBs/processRequest.completed                  470 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/default/processRequest.completed          462 ops
/myhost/OC4J:3001:6003/oc4j/default/WEBs/processRequest.completed                  451 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/JSPs/processRequest.completed  469 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/ojspdemos/processRequest.completed      452 ops
/myhost/OC4J:3001:6003/oc4j/ojspdemos/WEBs/processRequest.completed                472 ops
.
.
.
```

Using dmstool to Report All Metrics with Metric Values

Using `dmstool` with the `-dump` option displays all the metrics from an Oracle Application Server instance to the standard output.

The following command displays all available metrics:

```
% dmstool -dump
```

Oracle recommends that you run `dmstool` with the `-dump` option periodically, such as every 15 to 20 minutes, to capture and save a record of performance data. If you save performance data over time, this data can assist you if you need to analyze system behavior to improve performance or when problems occur.

Using dmstool to Report All Metrics with Metric Values in XML Format

When you need to process metric data, use the `format=xml` query on the `dmstool` command line to report all metric values in XML format.

The following command displays all available metrics using XML format:

```
% dmstool -dump format=xml
```

Using dmstool to View Metrics on a Remote Oracle Application Server System

Using `dmstool` with the `-a` option reports the metrics from a remote Oracle Application Server instance.

Note: By default the Oracle Application Server only allows `dmstool` to access metrics from the localhost. You need to modify `dms.conf` to support access from systems other than the localhost. See "AggreSpy URL and Access Control" on page 2-8 for information on DMS access control.

The following command displays all available metrics and metric values on the Oracle Application Server Instance, as specified with the `–a` option:

```
% dmstool –a opmn://system1:6003 -list
```

Using the `dmstool` -a option, supply an argument with the prefix `opmn://` and include the host name where you want to obtain metrics, and the OPMN request port number. The port specifies the OPMN request port that supplies metrics for Oracle Application Server which is specified the `request` attribute under the `<notification-server>` element in `$ORACLE_HOME/opmn/conf/opmn.xml` on UNIX and `%ORACLE_HOME%\opmn\conf\opmn.xml` on Windows.

See Also: "AggreSpy URL and Access Control" on page 2-8

Viewing Performance Metrics Using Spy (for Standalone OC4J)

When you are using OC4J in standalone mode, without the Oracle Application Server, the `Spy` Servlet allows you to access OC4J metrics in a manner similar to `AggreSpy`.

When running OC4J standalone, access performance metrics using `Spy` from the following URL:

```
http://myhost:myport/dmsoc4j/Spy
```

Note: You can only run `Spy` when OC4J is configured to support it, and OC4J is running. By default, OC4J supports `Spy`.

Table 2–3 covers the dmstool option that only applies to OC4J standalone mode. In addition, the options shown in Table 2–2 also apply to dmstool (except the -a option with the `opmn://` prefix.

Table 2–3 dmstool Command-line Options (for Standalone OC4J only)

Option	Description
`-a[ddress]` *`host`*`[:`*`port`*`][`*`path`*`],...`	For a standalone OC4J system, use the `-a` option. This specifies the http:// protocol, where: *`host`* is the domain name or IP address of the host on which the Oracle HTTP Server is running and *`port`* specifies the associated port.

3

Monitoring Oracle HTTP Server

This chapter discusses how to monitor Oracle HTTP Server performance. Obtaining performance data can assist you in tuning Oracle Application Server or in tuning and debugging applications with performance problems.

This chapter covers the following topics:

- Monitoring Oracle HTTP Server with Application Server Control
- Monitoring Oracle HTTP Server with Built-in Performance Metrics

Monitoring Oracle HTTP Server with Application Server Control

The Oracle HTTP Server is a central and important part of most Oracle Application Server sites. Oracle HTTP Server handles nearly every request for dynamic data and many static data requests as well. By monitoring Oracle HTTP Server performance you can identify and fix Oracle Application Server performance issues.

This section covers the following topics:

- Assessing the Oracle HTTP Server Load with Application Server Control
- Investigating Oracle HTTP Server Errors with Application Server Control
- Categorizing Oracle HTTP Server Problems with Application Server Control

Assessing the Oracle HTTP Server Load with Application Server Control

To monitor Oracle HTTP Server performance, the first step is to assess the workload (load).

When assessing the Oracle HTTP Server load, note the following:

- If you are developing or testing a new application, you need to determine how much load your quality assurance and performance tests generate on Oracle HTTP Server.
- If you are monitoring Oracle HTTP Server performance, note that usage often fluctuates depending on the time of day or day of week, with sites experiencing times with light loads, and times with heavy loads. Your performance tests and performance baseline should take into account the effect of time of day and day of week variances. Whether you are developing or administering an Oracle Application Server site, you should always design for expected load ranges and monitor the site to ensure that usage and performance remains within the expected range. You can use dmstool for periodic system monitoring.
- The Oracle HTTP Server performance information provides a picture of overall site performance; however if Oracle Application Server Web Cache or other caching mechanisms handle requests before they reach Oracle HTTP Server, then you need to monitor the caches as well.

Application Server Control provides Oracle HTTP Server performance data in the following categories:

- Oracle HTTP Server Status Metrics
- Oracle HTTP Server Response and Load Metrics
- Oracle HTTP Server Module Metrics
- Oracle HTTP Server Error Log

See Also:

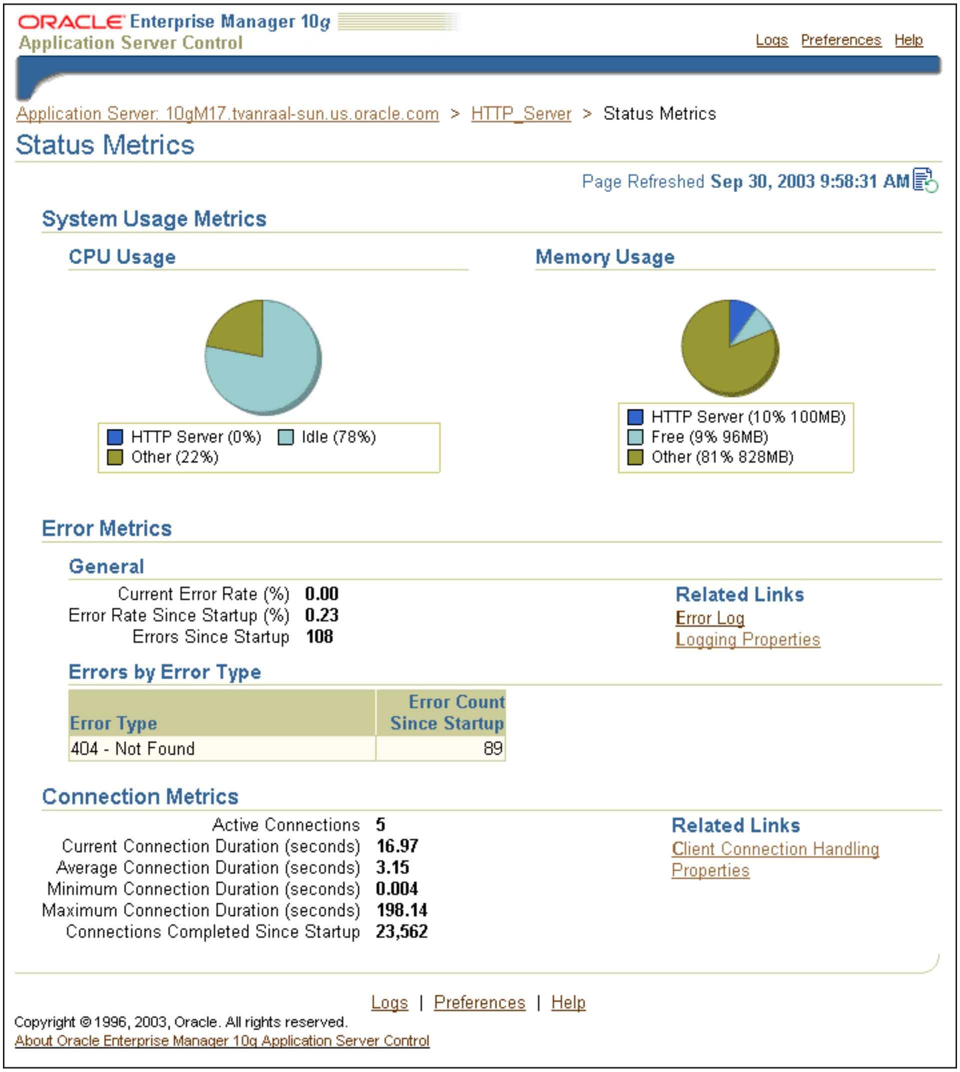
- "Performance Methodology" on page 1-8
- "Viewing Performance Metrics Using dmstool" on page 2-10
- Chapter 7, "Optimizing OracleAS Web Cache"
- *Oracle Application Server Web Cache Administrator's Guide* for further details on Oracle Application Server Web Cache
- *Oracle Application Server 10g Administrator's Guide* for information on using Application Server Control

Oracle HTTP Server Status Metrics

The Application Server Control status metrics provide information on CPU usage, memory usage, Oracle HTTP Server errors, and the number of active connections.

Figure 3–1 shows the Application Server Control HTTP Server status metrics page.

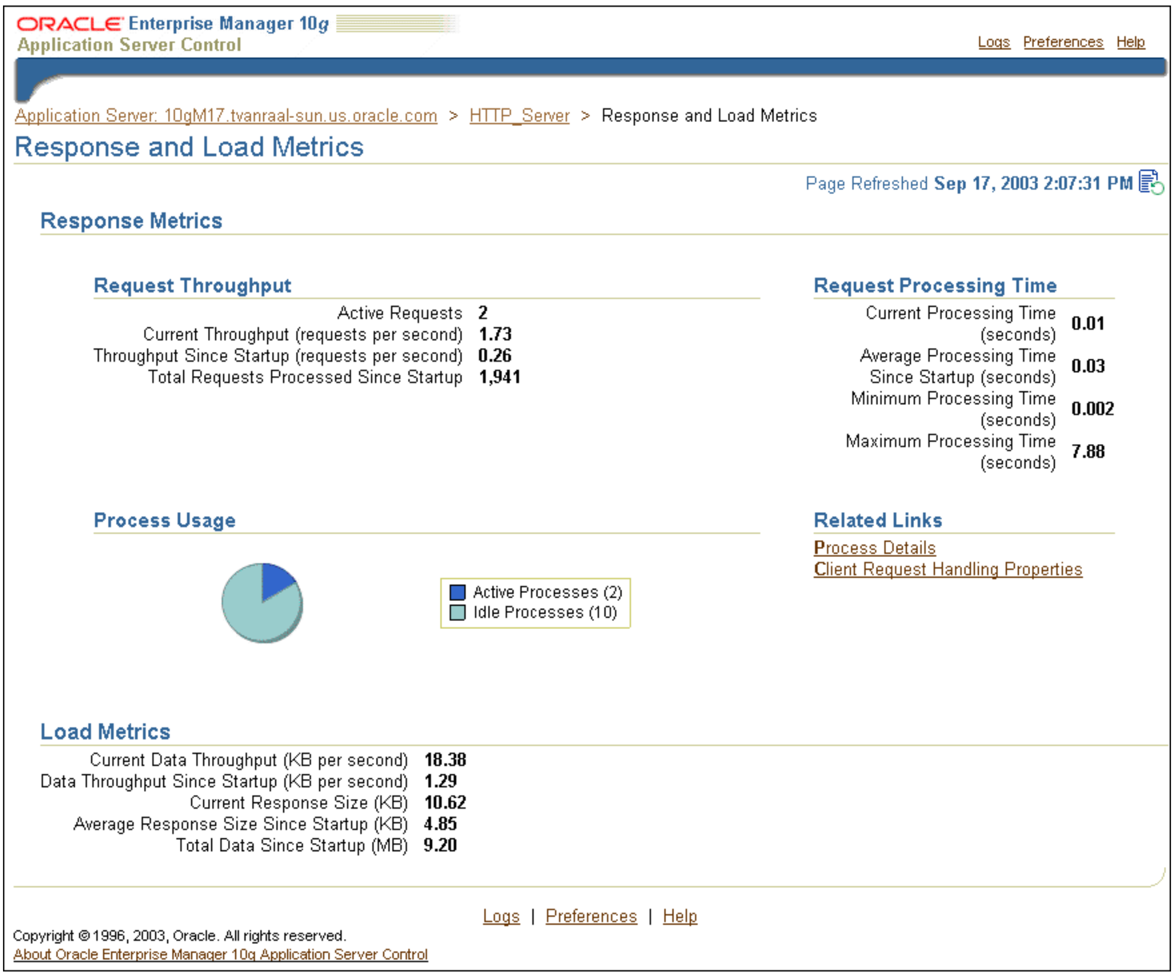
Figure 3–1 Application Server Control Status Metrics Page



Oracle HTTP Server Response and Load Metrics

Figure 3–2 shows the Application Server Control Response and Load Metrics page. This page shows values for Oracle HTTP Server Active Requests and Request Throughput, and reports the average, minimum, and maximum processing time for requests. The values on the Response and Load Metrics page can help you assess the system load.

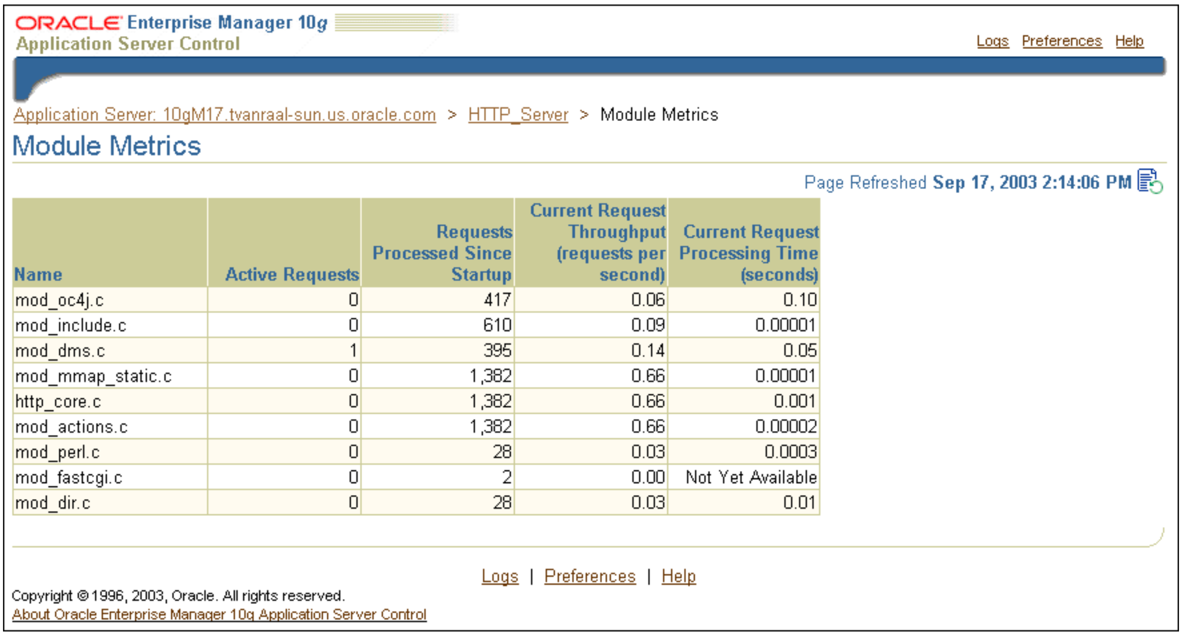
Figure 3–2 Application Server Control Response and Load Metrics



Oracle HTTP Server Module Metrics

Figure 3–3 shows the Application Server Control Module Metrics page. The Module Metrics page shows the active and total requests processed by Oracle HTTP Server modules. The page only lists modules active since startup, meaning that the module has received 1 or more requests.

Figure 3–3 Application Server Control Module Metrics Page



Name	Active Requests	Requests Processed Since Startup	Current Request Throughput (requests per second)	Current Request Processing Time (seconds)
mod_oc4j.c	0	417	0.06	0.10
mod_include.c	0	610	0.09	0.00001
mod_dms.c	1	395	0.14	0.05
mod_mmap_static.c	0	1,382	0.66	0.00001
http_core.c	0	1,382	0.66	0.001
mod_actions.c	0	1,382	0.66	0.00002
mod_perl.c	0	28	0.03	0.0003
mod_fastcgi.c	0	2	0.00	Not Yet Available
mod_dir.c	0	28	0.03	0.01

Oracle HTTP Server Error Log

The Error Log link displays the Application Server Control View Logs page and selects the HTTP Server logs.

> **See Also:** *Oracle Application Server 10g Administrator's Guide* for information on working with the View Logs page

Investigating Oracle HTTP Server Errors with Application Server Control

You should thoroughly investigate Oracle HTTP Server errors occurring on your site. Oracle HTTP Server errors may indicate acceptable activity, but they may also indicate security problems, configuration errors, or application bugs. Errors almost always affect Oracle Application Server performance. Error handling can slow down the normal processing for requests, or can appear to improve performance when the error handling abbreviates the processing required to handle a valid request.

Using Application Server Control you can view the Error Metrics on the HTTP Status Metrics page, as shown in Figure 3–1. Error Metrics include the current error rate, which is the number of errors occurring in approximately the last five minutes as a percentage of the total requests, the error rate since startup, and the count of the total number of errors since startup. The Status Metrics page includes the Errors by Error Type table shown in Figure 3–1 which lists more details for HTTP errors, including the error types and error counts. This table breaks down each error into a category based on its HTTP error response type.

The data values shown for Errors by Error Type in Figure 3–1 indicate that the errors were due to requests for unknown URIs (`404 - Not Found` errors). On many Oracle HTTP Server sites, Not Found errors are relatively common. However, you should investigate reports showing large numbers of Not Found errors, such as a number that is greater than 1% of the total requests (see Figure 3–2 to find the total requests processed in the Request Throughput area on the Response and Load Metrics page).

To investigate errors in more detail, such as any reported internal errors, examine the error log by selecting the Logs link from any page, or the Error Log link under the Related Links heading on the Status Metrics page. By examining the error log file entries, you should be able to find more information about the URIs that are causing specific errors.

> **See Also:** *Oracle Application Server 10g Administrator's Guide* for information on working with the View Logs page

Expected Oracle HTTP Server Errors and Warnings

Certain Oracle HTTP Server errors and warnings are expected during normal Oracle Application Server operations. For example, errors and warnings occur when the OC4J instance is stopped or restarted when you perform certain configuration actions using Application Server Control.

Example 3–1 shows some of the types of errors that you may see during an OC4J restart operation.

Example 3–1 Expected Errors Occurring During OC4J Restart Operation

```
MOD_OC4J_0150: Failed to deterministicly find a failover oc4j process for session request for
island: default_island for destination: home.
MOD_OC4J_0119: Failed to get an oc4j process for destination: home
MOD_OC4J_0013: Failed to call destination: home's service() to service the request.
MOD_OC4J_0150: Failed to deterministicly find a failover oc4j process for session request for
island: default_island for destination: home.
.
.
.
MOD_OC4J_0119: Failed to get an oc4j process for destination: home
MOD_OC4J_0013: Failed to call destination: home's service() to service the request.
MOD_OC4J_0150: Failed to deterministicly find a failover oc4j process for session request for
island: default_island for destination: home.
MOD_OC4J_0119: Failed to get an oc4j process for destination: home
MOD_OC4J_0013: Failed to call destination: home's service() to service the request.
(131)Connection reset by peer: MOD_OC4J_0086: Got an unexpected error while calling recv() to
receive a message from oc4j and error code is 131.
MOD_OC4J_0054: Failed to call network routine to receive an ajp13 message from oc4j.
MOD_OC4J_0033: Failed to receive an ajp13 message from oc4j.
(131)Connection reset by peer: MOD_OC4J_0086: Got an unexpected error while calling recv() to
receive a message from oc4j and error code is 131.
MOD_OC4J_0054: Failed to call network routine to receive an ajp13 message from oc4j.
MOD_OC4J_0033: Failed to receive an ajp13 message from oc4j.
```

Categorizing Oracle HTTP Server Problems with Application Server Control

If you notice a performance problem on the Oracle HTTP Server, then if possible you should drill down and categorize the problem. By refining the performance analysis you can learn more about the issue and direct your efforts to a component to help identify and resolve the problem.

Application Server Control can help you categorize performance problems. You can identify where requests are being processed, or where a large percentage of request processing time is concentrated. Using Application Server Control allows you to categorize performance problems as follows:

- Categorizing Oracle HTTP Server Problems by Module
- Categorizing Oracle HTTP Server Problems by Virtual Host
- Categorizing Oracle HTTP Server Problems by Child Server

Categorizing Oracle HTTP Server Problems by Module

Figure 3–3 shows the Module Metrics for Oracle HTTP Server modules (the report includes information for modules that have received 1 or more requests since startup). Using the Module Metrics, you should be able to identify the name of the module that processed a large number of requests, or identify a module where the processing time for an individual request is very large. By looking at the values for metrics listed in the Module Metrics table, you should be able to categorize Oracle Application Server performance by module.

When viewing the Module Metrics, note the following:

1. The `http_core.c` module handles every request for static pages. If Oracle Application Server Web Cache is enabled, then use of `http_core.c` should be reduced. When you are using Oracle Application Server Web Cache, you should monitor requests processed by the `http_core.c` module to make sure that Oracle Application Server Web Cache effectively reduces static page activity for the Oracle HTTP Server.

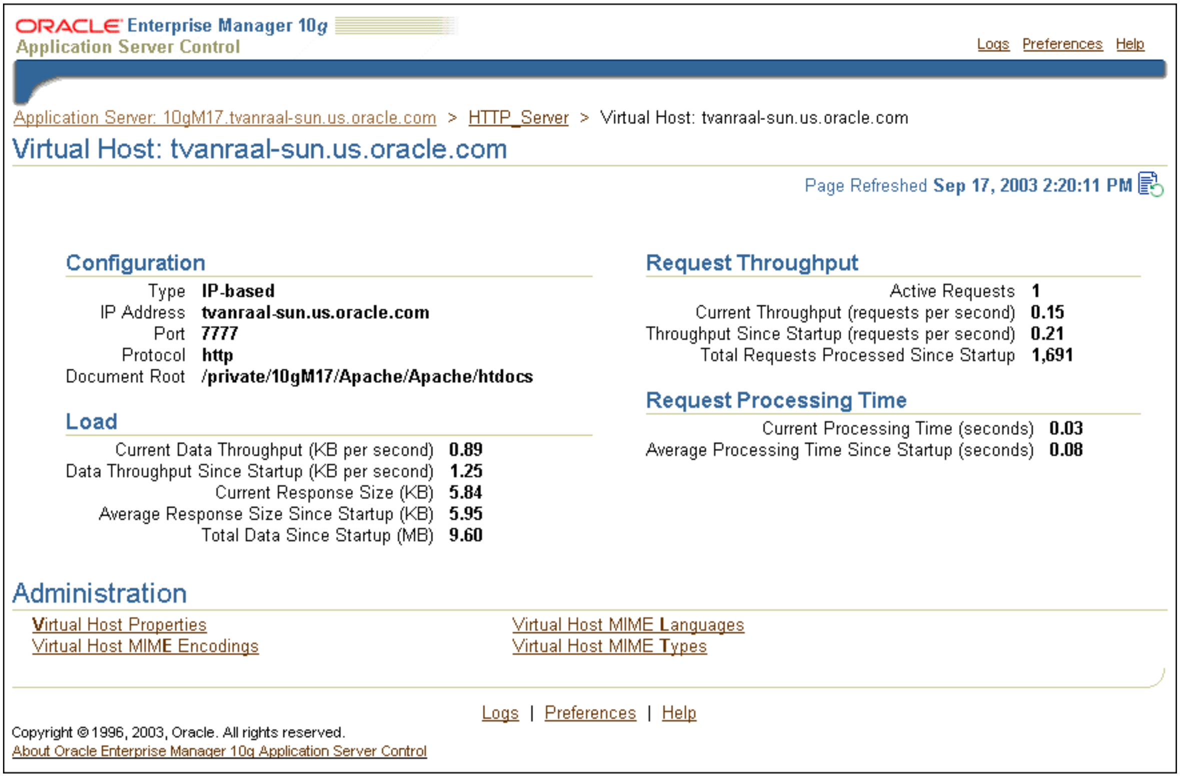
2. Viewing the Module Metrics page may show you that many requests were processed through the `mod_oc4j.c` module. You should then drill down to review the information available for your OC4J instances. Application Server Control provides extensive performance measurements for OC4J instances and J2EE applications.

   See Also: Chapter 4, "Monitoring OC4J"

Categorizing Oracle HTTP Server Problems by Virtual Host

Figure 3–4 shows a display of the Virtual Host page. By viewing the Virtual Host page you should be able to obtain information about request processing by virtual host. The Request Throughput, Load, and Request Processing Time headings include information that enables you to identify a virtual host on your system that is processing a large number of requests, or that is using significant processing resources and may be stressing the system. This information should help you to categorize Oracle Application Server performance issues by virtual host.

Figure 3–4 Application Server Control Virtual Host Page



Categorizing Oracle HTTP Server Problems by Child Server

Running Oracle HTTP Server, usually you do not need to worry about which child server handles an individual request because any available child server can handle any incoming request (each request is handled by a free child server). However, if your Oracle Application Server system experiences delays or deadlocks, you may need to analyze the Oracle HTTP Server child server processes.

To obtain information on Oracle HTTP Server child server processes, select Response and Load Metrics link from the HTTP Server page, and then, under Related Links, select Process Details. The Process Details page shows the Process ID for each active Oracle HTTP Server child process.

Viewing the Process Details page allows you to monitor child servers to identify runtime problems, configuration errors, or application bugs that cause either request processing deadlocks or very long delays. In these situations analyzing the Process Details page can help determine where the deadlock or delay is occurring.

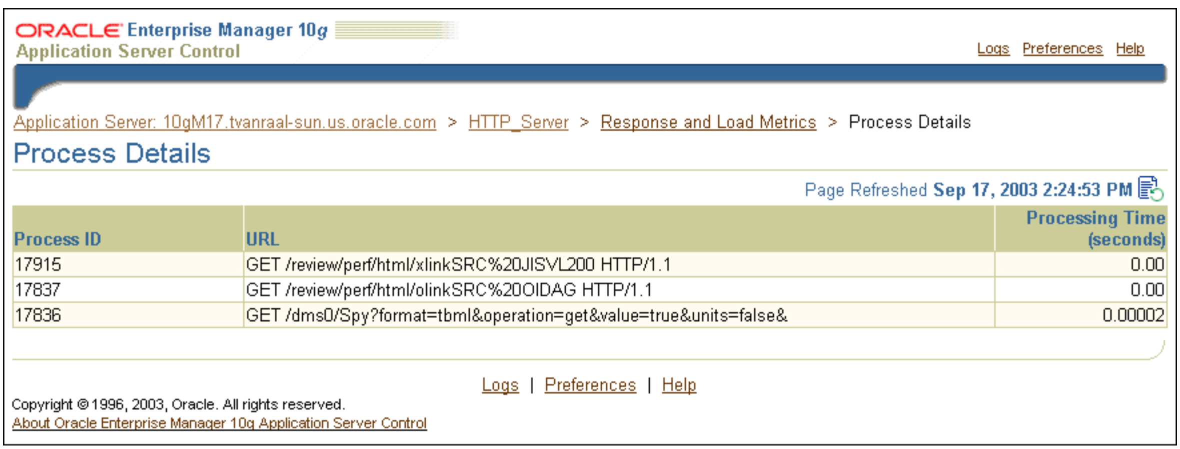
Figure 3–5 shows a Process Details page with Oracle HTTP Server child server information.

When viewing the Oracle HTTP Server Process Details page, note the following:

1. If necessary you can use the Process ID value to identify and terminate a deadlocked Oracle HTTP Server child server.
2. Oracle HTTP Server terminates requests after a configurable timeout. You can use Application Server Control to set the timeout for requests.

> **See Also:** *Oracle HTTP Server Administrator's Guide* for information on the `TimeOut` directive

Figure 3–5 Application Server Control HTTP Server Process Details for Child Servers Page



Monitoring Oracle HTTP Server with Built-in Performance Metrics

The Oracle HTTP Server is a central and important part of most Oracle Application Server sites. Oracle HTTP Server handles nearly every request for dynamic data and many static data requests as well. By monitoring Oracle HTTP Server performance, you can identify and fix Oracle Application Server performance issues.

This section covers the following topics:

- Assessing the Oracle HTTP Server Load with Built-in Metrics
- Investigating Oracle HTTP Server Errors with Built-in Metrics
- Categorizing Oracle HTTP Server Performance Problems with Built-in Metrics

Assessing the Oracle HTTP Server Load with Built-in Metrics

To monitor Oracle HTTP Server performance, the first step is to assess workload.

When assessing the Oracle HTTP Server workload (load), note the following:

- If you are developing or testing a new application, you need to determine how much load your quality assurance and performance tests generate on Oracle HTTP Server.
- If you are monitoring Oracle HTTP Server performance, note that usage often fluctuates depending on the time of day or day of week, with sites experiencing times with light loads, and times with heavy loads. Your performance tests and performance baseline should take into account the effect of time of day and day of week variances. Whether you are developing or administering an Oracle Application Server site, you should always design for expected load ranges and monitor the site to ensure that usage and performance remains within the expected range.
- The Oracle HTTP Server performance metrics give a good picture of overall site performance; however if Oracle Application Server Web Cache or other caching mechanisms handle requests before they reach Oracle HTTP Server, then you need to monitor the caches as well.

See Also: "Performance Methodology" on page 1-8

Oracle HTTP Server provides performance metrics which you can view using `AggreSpy` or `dmstool`. You can use these built-in performance tools to help you assess Oracle HTTP Server load by viewing the `ohs_server` metric table. Using `AggreSpy` or `dmstool`, you can view the `ohs_server` metric table.

Example 3–2 shows the `dmstool` command with the `ohs_server` metrics output. You can also view the `ohs_server` metric table using `AggreSpy` by choosing the `ohs_server` metric table in the left pane of the `AggreSpy` window or by selecting the `Text` link next to the `Apache` process in the `AggreSpy` All DMS Spies list. If you select the `Apache` process from the Spies List, you need to find the `ohs_server` table within the full set of metrics shown.

Example 3–2 Overall HTTP Server Metrics Report

```
system1 122> dmstool -table ohs_server
Fri May 02 11:11:39 PDT 2003
----------
ohs_server
----------
busyChildren.value:     1
childFinish.count:      0       ops
childStart.count:      11       ops
connection.active:3 threads
connection.avg:258721053 usecs
connection.completed:   11880   ops
connection.maxTime:1002008298 usecs
connection.minTime:7254 usecs
connection.time:152386700540 usecs
error.count:    52       ops
get.count:      32769   ops
handle.active:2 threads
handle.avg:14274 usecs
handle.completed:6985
handle.maxTime:22205524 usecs
handle.minTime:2 usecs
handle.time:997159521 usecs
internalRedirect.count: 7418   ops
lastConfigChange.value: 1051724112
numChildren.value:      11
numMods.value:  47
post.count:     0       ops
readyChildren.value:    10
request.active: 1       threads
request.avg:31537 usecs
request.completed:32769
request.maxTime:22206941 usecs
request.minTime:602 usecs
request.time:1033442848 usecs
responseSize.value:     243880796
Host:   system1
Name:   Apache
Parent: /
Process:        Apache:27885:6004
```

First, to analyze system performance, the output shown in Example 3–2 includes three categories of metrics to be inspected: `handle`, `request`, and `connection`. These metrics describe the following:

- `handle`

  The phase in which a request is handled by an HTTP server module. Note that a single request may be handled by more than one HTTP server module. The handle metrics shown in the `ohs_server` metric table are summarized for all of the HTTP server modules.

- `request`

  The phase during which an HTTP server daemon reads a request and sends a response for it (first byte in, last byte out). There may be more than one request serviced during a single connection phase. This would be the case if the HTTP parameter `KeepAlive` were set and utilized by clients.

- `connection`

  The connection phase, starting from the time an HTTP connection is established to the time it is closed.

Second, to determine current Oracle HTTP Server load, examine the following `ohs_server` metrics:

- `request.active`
- `busyChildren.value`
- `readyChildren.value`
- `numChildren.value`

These `ohs_server` metrics indicate how many OHS child servers, children, are in use, and how many of child servers are actively processing requests. The data in Example 3–2 shows that 11 child servers are alive (`numChildren.value`), one of which is currently busy handling requests (`busyChildren.value`).

Oracle HTTP Server needs to keep enough child servers running to handle the usual load while allowing for normal load fluctuations. Oracle HTTP Server child servers handle exactly one request at a time, thus Oracle HTTP Server needs to run many child servers at once. If Oracle HTTP Server notices that the current load may exceed its default configuration, then it automatically starts new child servers. If the load is subsequently reduced, then Oracle HTTP Server terminates some of its child servers to save system resources.

If the configuration settings require that the Oracle HTTP Server start and stop child servers frequently, this can reduce system performance and may indicate that the system configuration needs to be adjusted. To determine whether Oracle HTTP Server child servers have been started and how many have finished, examine the following `ohs_server` metrics:

- `childStart.count`
- `childFinish.count`

These performance metrics show how many Oracle HTTP Server child servers have started and finished and can also provide an indication of the Oracle HTTP Server load. For the Oracle HTTP Server shown in Example 3–2, 11 child servers have started and 0 have finished.

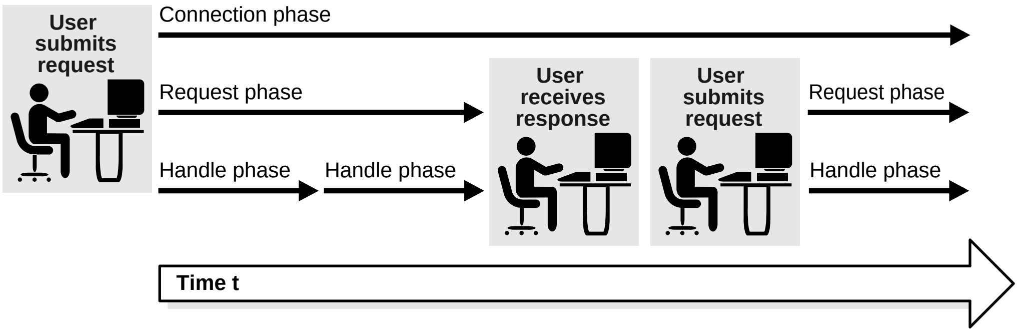
The `childStart.count` and `childFinish.count` metric values could indicate that the instantaneous load for the Oracle HTTP Server exceeded the current load and also exceeded the range assumed by the default Oracle HTTP Server configuration parameters. When the count of child servers started and the count of child servers finished are both large, this could indicate that the Oracle HTTP Server could benefit by tuning configuration parameters, including:

- `MinSpareServers`
- `MaxSpareServers`
- `StartServers`

In the `ohs_server` metrics, the `handle.avg`, `request.avg`, and `connection.avg` metrics, and the `handle.time`, `request.time`, and `connection.time` values increase for each phase. The handle time will be the shortest and the connection time the longest. Figure 3–6 shows the relationship among these three phases for managing a user request.

If `KeepAlive` is on and clients use it, the duration of a connection may be much longer than the time required to perform a request and return a response, as illustrated in Figure 3–6. This is because the connection may remain open while a single client submits multiple requests.

Figure 3–6 Execution Phases in the Oracle HTTP Server



See Also:

- Chapter 5, "Optimizing Oracle HTTP Server"
- Chapter 7, "Optimizing OracleAS Web Cache"
- Appendix A, "Performance Metrics"
- *Oracle Application Server Web Cache Administrator's Guide*
- *Oracle HTTP Server Administrator's Guide* for information on Oracle HTTP Server configuration parameters related to starting and stopping child servers

Investigating Oracle HTTP Server Errors with Built-in Metrics

You should thoroughly investigate Oracle HTTP Server errors occurring on your site. Oracle HTTP Server errors may indicate acceptable activity, but they may also indicate security problems, configuration errors, or application bugs. Errors almost always affect Oracle Application Server performance. Error handling can slow down the normal processing for requests, or can appear to improve performance when the error handling abbreviates the processing required to handle a valid request.

Using `dmstool` or `AggreSpy`, you can investigate Oracle HTTP Server errors by viewing the `ohs_server` metrics. Example 3–2 includes the `ohs_server` metrics that provide an overview of error activity. The `error.count` metric increments whenever any request to Oracle HTTP Server results in an HTTP error response.

Use the `ohs_responses` metric table to investigate the details for error types and error counts. This table breaks down the total `error.count` value into HTTP response types. It also shows aggregate counts for successful HTTP requests and HTTP redirects.

Example 3–3 shows the `dmstool` report for the `ohs_responses` metric table. You can also view the `ohs_responses` metric table using `AggreSpy` by choosing the `ohs_responses` metric table in the left pane of the `AggreSpy` window or by selecting the `Text` link next to the `Apache` process in All DMS Spies list. If you select the Apache process from the Spies List, you need to find the `ohs_responses` table within the full set of metrics shown.

Example 3–3 HTTP Server Responses Metrics (ohs_responses Metric Table)

```
system1 125> dmstool -table ohs_responses

Fri May 02 15:19:56 PDT 2003
-------------
ohs_responses
-------------
CltErr_Authorization_Required_401.count:        0       ops
CltErr_BadRange_416.count:      0       ops
.
.
.
CltErr_Not_Found_404.count:     29      ops
.
.
.
Redirect_MultiChoice_300.count: 0       ops
Redirect_NotModified_304.count: 23      ops
Success_Accepted_202.count:     0       ops
.
.
.
SvrErr_VersionNotSupp_505.count:        0       ops
Host:   system1
Name:   Responses
Parent: /Apache
Process:        Apache:27885:6004
ohs_server:     Apache
```

Example 3–3 shows that most of the errors were due to requests for unknown URIs (`404 - Not Found` errors). On many Oracle HTTP Server sites, Not Found errors are relatively common. However, you should investigate reports showing many Not Found errors, such as a number greater than 1% of the total requests.

You can examine the `error_log` and `access_log` files to determine the URIs that are causing errors, such as any reported internal errors (`SvrErr_InternalError_500.count`).

See Also:

- "Expected Oracle HTTP Server Errors and Warnings" on page 3-7
- *Oracle HTTP Server Administrator's Guide* for information on the Oracle HTTP Server `access_log` and `error_log` files
- *Oracle Application Server 10g Administrator's Guide* for information on working with the View Logs page.

Categorizing Oracle HTTP Server Performance Problems with Built-in Metrics

If you notice a performance problem on the Oracle HTTP Server, then if possible you should drill down and categorize the problem. By limiting your search for a performance problem to a subset of Oracle HTTP Server, you can learn more about the issue and direct your efforts to identifying and solving the problem. Using the built-in performance tools you can categorize performance problems into one of several areas. You can identify where requests are being processed, or where a large percentage of request processing time is concentrated.

This section describes how you can categorize performance problems into different areas, including:

- Categorizing Oracle HTTP Server Performance Problems by Module
- Categorizing Oracle HTTP Server Performance Problems by Virtual Host
- Categorizing Oracle HTTP Server Performance Problems by Child Server

Categorizing Oracle HTTP Server Performance Problems by Module

Use the `ohs_module` metrics to refine your analysis of performance problems to one or more modules. Showing the module metrics allows you to use the metric data to limit the search for performance problems to a particular module.

Example 3–4 shows a section of the `dmstool` report for the `ohs_module` metric table. You can also view the `ohs_module` metric table using `AggreSpy` by choosing the `ohs_module` link in the left pane of the `AggreSpy` window or by selecting the `Text` link next to the `Apache` process in the All DMS Spies list. If you select the Apache process from the Spies List, you need to find the `ohs_module` table within the full set of metrics shown.

Example 3–4 Drill Down to Investigate Oracle HTTP Server Activity per Module

```
system1 127> dmstool -table ohs_module -c 1
Fri May 02 15:51:01 PDT 2003
----------
ohs_module
----------
decline.count:  76661   ops
handle.active:  0       threads
handle.avg:     13      usecs
handle.completed:       76661   ops
handle.maxTime: 5487    usecs
handle.minTime: 11      usecs
handle.time:    1007639 usecs
Host:   system1
Name:   mod_actions.c
Parent: /Apache/Modules
Process:        Apache:27885:6004
ohs_server:     Apache
.
.
.
Name:   mod_plsql.c
.
.
.
decline.count:  0       ops
handle.active:  0       threads
handle.avg:     919     usecs
handle.completed:       76708   ops
handle.maxTime: 122401  usecs
handle.minTime: 351     usecs
handle.time:    70532228        usecs
Host:   system1
Name:   http_core.c
Parent: /Apache/Modules
Process:        Apache:27885:6004
ohs_server:     Apache
.
.
```

```
.
decline.count:  0       ops
handle.active:  0       threads
handle.avg:     331918  usecs
handle.completed:       440     ops
handle.maxTime: 42707927        usecs
handle.minTime: 5970    usecs
handle.time:    146044090       usecs
Host:   system1
Name:   mod_oc4j.c
Parent: /Apache/Modules
Process:        Apache:27885:6004
ohs_server:     Apache
```

When viewing the Module Metrics, note the following:

1. The `http_core.c` module handles every request for static pages. If Oracle Application Server Web Cache is enabled, then use of `http_core.c` should be reduced. If Oracle Application Server Web Cache is enabled then you should monitor the `http_core.c` metrics to make sure that Oracle Application Server Web Cache effectively prevents static page activity from reaching your Oracle HTTP Server.

2. Typically, certain responses require process initialization, class loading or other one-time processing that can skew the reporting of the average request processing time. For performance reporting and analysis, you can reduce the effect of the such one-time operations by subtracting the minimum and maximum values from the total and recalculating the average. For example, for the `mod_oc4j.c` metrics shown in Example 3–4, if you recompute the request handling average using the following formula, you find that the recalculated average provides a more representative indication of typical response processing time:

   ```
   new average = (time - min - max) / (completed - 2)
               = (146044090 - 5970 - 42707927)/ (440 - 2)
               =   305710.6 microseconds
   ```

   Recalculating the average is most important when the server has been up for a short time, and thus has handled a small number of requests. In this case, the large overhead of the first request will have far more impact on the average.

3. Viewing the `ohs_module` metric table may show you that many requests were forwarded to OC4J through the `mod_oc4j.c` module. Oracle Application Server also provides extensive performance measurements for OC4J J2EE applications.

   See Also: Chapter 4, "Monitoring OC4J"

Categorizing Oracle HTTP Server Performance Problems by Virtual Host

Use the `ohs_virtualHost` metrics to refine your analysis of performance problems by Oracle HTTP Server virtual host. Showing the virtual host metrics allows you to use the metric data to limit the search for performance problems to a subset of the Oracle HTTP Server.

Example 3–5 shows a section of the `dmstool` report for the `ohs_virtualHost` metric table. You can also view the `ohs_virtualHost` metric table using `AggreSpy` by choosing the `ohs_virtualHost` link in the left pane of the `AggreSpy` window or by selecting the `Text` link next to the `Apache` process in the All DMS Spies list. If you select the Apache process from the Spies List, you need to find the `ohs_virtualHost` table within the full set of metrics shown.

Example 3–5 Drill Down to Investigate Oracle HTTP Server Activity per Virtual Host

```
system1 134> dmstool -table ohs_virtualHost -c 1
Mon May 05 10:35:10 PDT 2003
---------------
ohs_virtualHost
---------------
request.active: 0       threads
request.avg:    0       usecs
request.completed:      0       ops
request.maxTime:        0       usecs
request.minTime:        0       usecs
request.time:   0       usecs
responseSize.value:     0       bytes
vhostType.value:        IP_DEFAULT
Host:   system1
Name:   system1.us.oracle.com:IP255.255.255.255,Port4444
Parent: /Apache/VHosts
Process:        Apache:27885:6004
ohs_server:     Apache
ohs_vhostSet:   VHosts
```

Categorizing Oracle HTTP Server Performance Problems by Child Server

Running Oracle HTTP Server, usually you do not need to worry about which child server handles an individual request because any available child server can handle any incoming request (each request is handled by a free child server). However, if your Oracle Application Server system experiences delays or deadlocks, you may need to analyze the Oracle HTTP Server child server metrics. These metrics allow you to monitor child servers to identify runtime problems, configuration errors, or application bugs that cause either request processing deadlocks or very long delays. In these situations analyzing the Oracle HTTP Server child server metrics can help determine where the deadlock or delay is occurring.

Use the `ohs_child` metric table to refine your analysis of performance problems to one or more Oracle HTTP Server child servers.

Example 3–6 shows a section of the `dmstool` report for the `ohs_child` metric table. You can also view the `ohs_child` metric table using `AggreSpy` by choosing the `ohs_child` link in the left pane of the `AggreSpy` window or by selecting the `Text` link next to the `Apache` process in the All DMS Spies list. If you select the Apache process from the Spies List, you need to find the `ohs_child` table within the full set of metrics shown

The `ohs_child` metric table shows the top ten Oracle HTTP Server child servers sorted by time spent on current requests. For the metrics shown in Example 3–6, the top entry has been executing for 7 microseconds. The `ohs_child` metrics include the URL associated with the request and the process identifier for each Oracle HTTP Server child server listed.

Example 3–6 Drill Down to Investigate Activity per Child Server

```
system1 135> dmstool -table ohs_child -c 1
Mon May 05 10:44:24 PDT 2003
userpasswd=null
---------
ohs_child
---------
.
.
.
pid.value:      27897
slot.value:     3
status.value:   writing
time.value:     1       usecs
url.value:      GET /dms0/Spy?format=tbml&operation=get&value=true&units=true&d
Host:   system1
```

```
Name:   Child01
Parent: /Apache/Children
Process:        Apache:27885:6004
ohs_server:     Apache
pid.value:      27899
slot.value:     5
status.value:   keepalive
time.value:     7       usecs
url.value:      GET /dmsDemo/BasicBinomial HTTP/1.1
Host:   system1
Name:   Child00
Parent: /Apache/Children
Process:        Apache:27885:6004
ohs_server:     Apache
```

When viewing the Oracle HTTP Server child server metrics, note the following:

1. If necessary you can use the `ohs_child` metric value `pid.value` to identify and terminate a deadlocked Oracle HTTP Server child server.
2. Oracle HTTP Server terminates requests after a configurable timeout set with the `TimeOut` directive.

   See Also: *Oracle HTTP Server Administrator's Guide* for information on the `TimeOut` directive

4 Monitoring OC4J

This chapter discusses how to monitor the performance of Oracle Application Server Containers for J2EE (OC4J). Obtaining performance data can assist you in tuning Oracle Application Server or in tuning and debugging applications with performance problems.

This chapter contains the following topics:

- Monitoring OC4J With Application Server Control
- Monitoring OC4J With Built-in Performance Metrics

Monitoring OC4J With Application Server Control

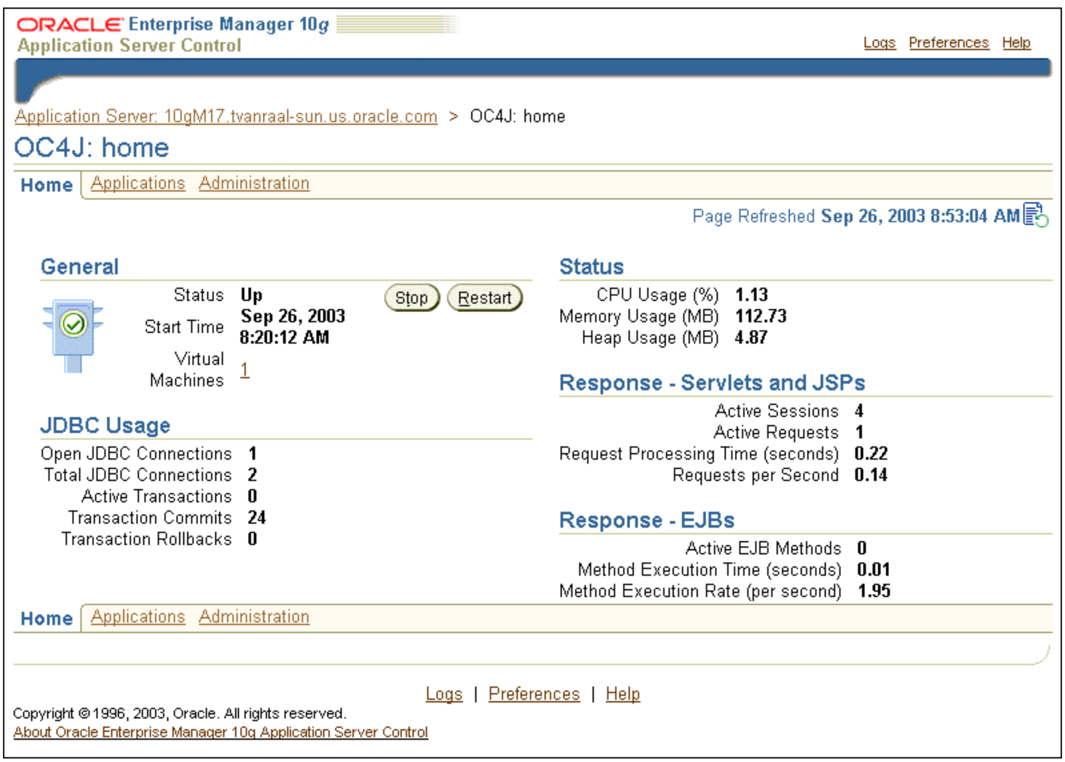
Using Application Server Control, you can view information on the performance characteristics of OC4J instances, J2EE applications, and Oracle Application Server components running under OC4J. This section covers the following:

- Monitoring OC4J Instances With Application Server Control
- Monitoring J2EE Applications with Application Server Control

Monitoring OC4J Instances With Application Server Control

Before analyzing OC4J performance, make sure that your OC4J instance is running. Figure 4–1 shows Application Server Control with the OC4J instance homepage. This page shows the status for a selected OC4J instance (the Up under the heading General indicates the OC4J instance is running).

Figure 4–1 Application Server Control OC4J Instance Display



Note: Application Server Control does not provide information on OC4J JMS. Use the built-in performance metrics to obtain information on OC4J JMS.

See Also: "Monitoring OC4J With Built-in Performance Metrics" on page 4-9

General

The Application Server Control OC4J General information provides information on up and down status for the OC4J instance, its start time, and information on the virtual machine where the OC4J instance is running. This area also presents buttons that allow you to stop or restart the OC4J instance.

JDBC Usage

The Application Server Control OC4J JDBC Usage information shows the number of open JDBC connections, the total number of JDBC connections, the number of active transactions, and the total number of transaction commits and transaction rollbacks for the OC4J instance.

Status

The Application Server Control OC4J Status information shows the CPU usage, memory usage, and heap usage for the OC4J instance.

Response for Servlets and JSPs

The Application Server Control OC4J Response information for Servlets and JSPs shows the number of active sessions, the active requests, the average request processing time, and the requests processed per second for active requests.The value shown for requests processed per second is a rate that is calculated using the requests processed over the previous 5 minutes.

Response for EJBs

The Application Server Control OC4J Response information for EJBs shows the number of active EJB methods and the EJB method execution rate. The EJB method execution rate provides the number of methods executed per second over the previous 5 minutes.

Note: Application Server Control automatically collects a subset of metrics approximately every five minutes. The rates shown in the Application Server Control display are computed over the period that spans from the most recent collection to the refresh of the Application Server Control display.

Monitoring J2EE Applications with Application Server Control

After you know that the OC4J instances that contain your J2EE applications are running, check the status for your applications. If your J2EE applications are not loaded, then deploy them and then try accessing the applications to make sure that they work properly. Figure 4–2 shows an Application Server Control page for the FAQApp sample application.

Figure 4–2 Application Server Control J2EE Application Metrics

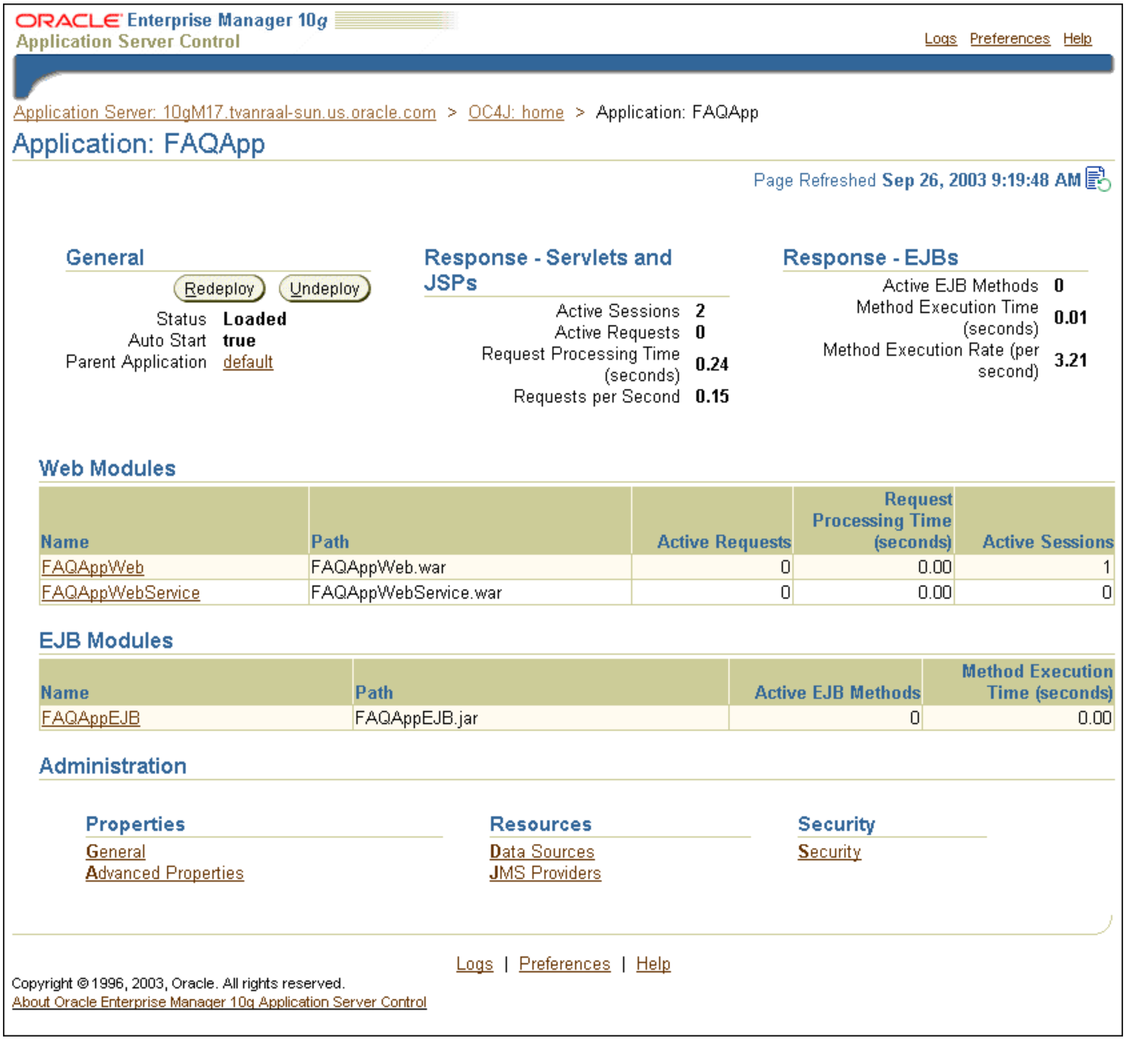


Figure 4–2 shows the available Application Server Control J2EE application level performance data collected in the following categories:

- General
- Response for Servlets and JSPs
- Response for EJBs
- Web Module Table
- EJB Modules Table

General

The Application Server Control J2EE application General information provides an indication of whether the application is loaded or not in the status field, and shows if the Auto Start status is true or false. The Parent Application field provides a link to the application parent. This area also presents buttons that allow you to Redeploy or Undeploy the application.

Response for Servlets and JSPs

The Application Server Control J2EE application Response information for Servlets and JSPs shows the number of active sessions, the active requests, the average request processing time, and the requests processed per second, over the previous 5 minutes, for active requests for the application. For more detail on this information or to drill down to specific Servlets and JSPs, use the links in the Web Modules table.

Response for EJBs

The Application Server Control J2EE application Response information for EJBs shows the number of active EJB methods and the EJB method execution rate over the previous 5 minutes.

For more detail on this information or to drill down to specific Servlets and JSPs, use the links in the EJB Modules table.

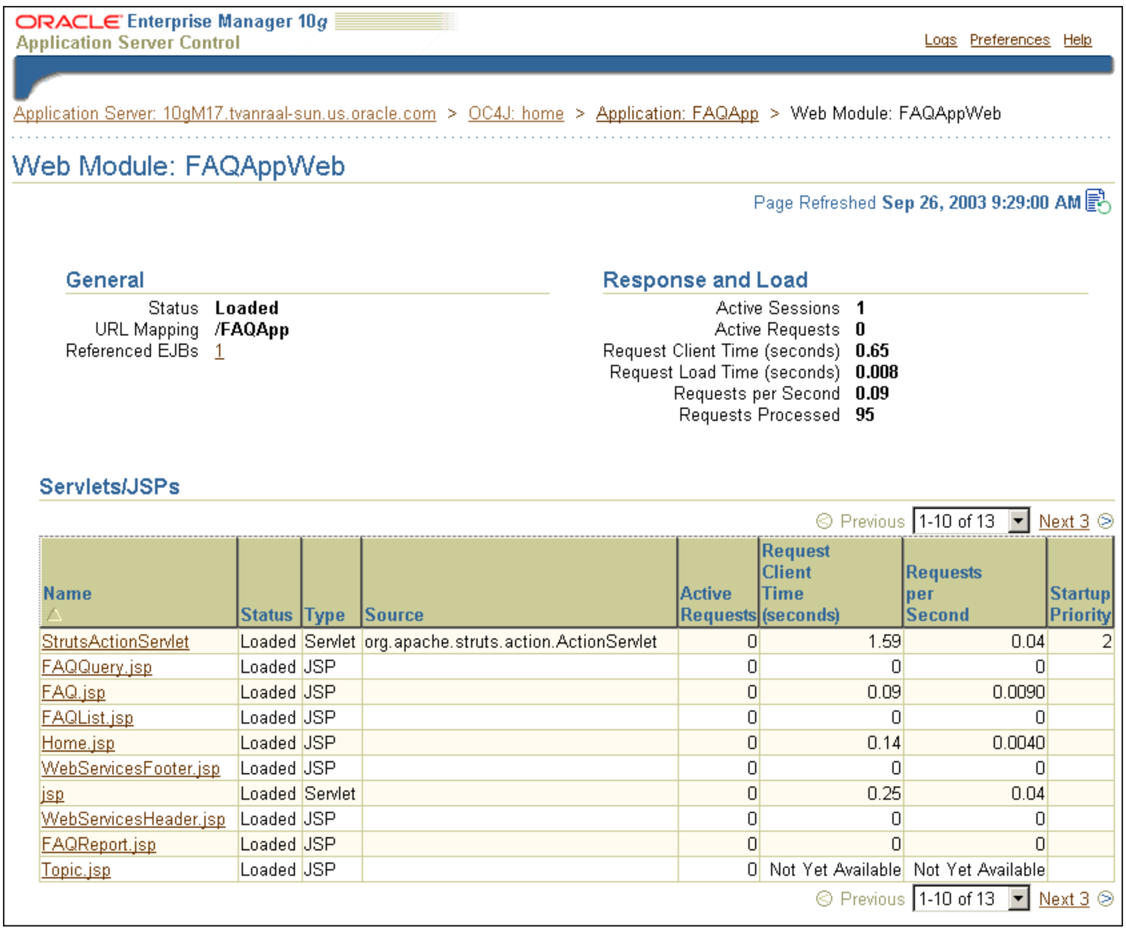
Note: Application Server Control automatically collects a subset of metrics approximately every five minutes. The rates shown in the Application Server Control display are computed over the period that spans from the most recent collection to the refresh of the Application Server Control display.

Web Module Table

The Web Modules table allows you to obtain more detailed information for Servlets and JSPs within a J2EE application.

Figure 4–3 shows the details for the FAQApp application's Web Module, including General information, Response and Load information, and a table showing data values for each of the Servlets and JSPs that are part of the application.

Figure 4–3 Application Server Control J2EE Application Web Module Metrics

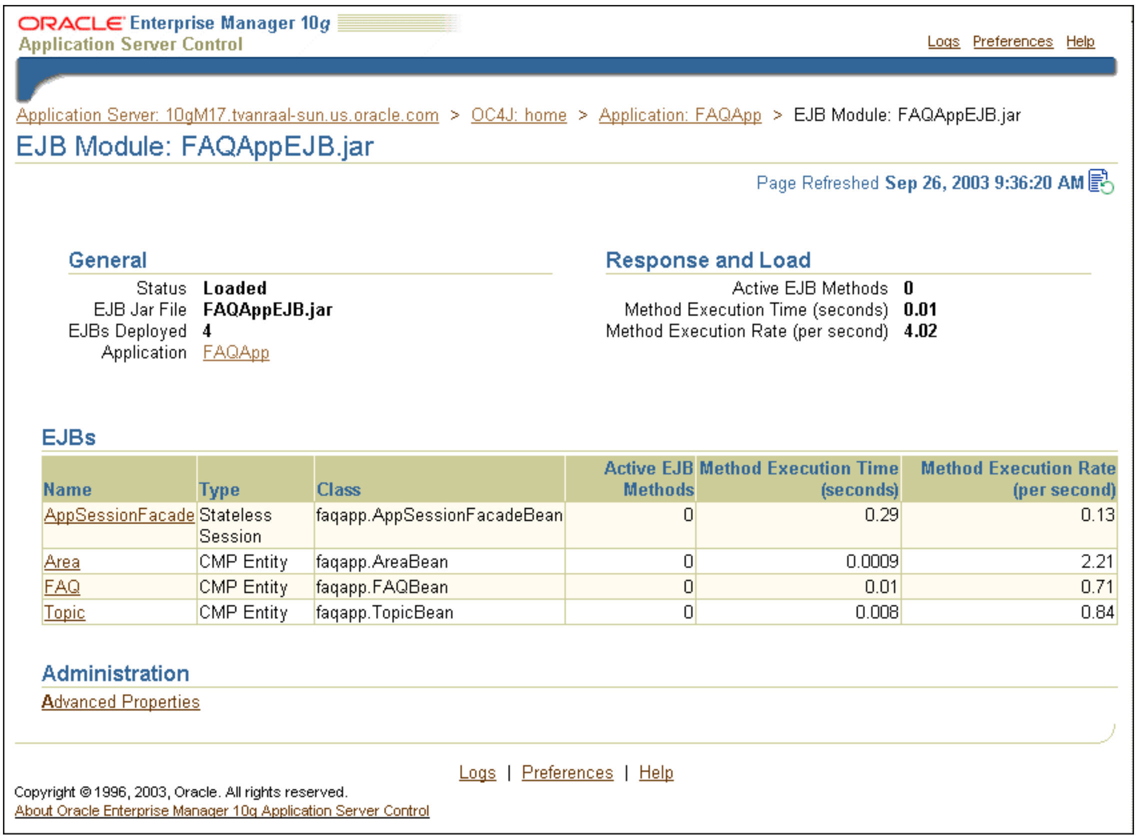


EJB Modules Table

The EJB Modules tables allow you to obtain more detailed information on EJB modules and EJBs within the J2EE application.

Figure 4–4 shows a sample FAQApp EJB Module page.

Figure 4–4 Application Server Control EJB Module Page



Monitoring OC4J With Built-in Performance Metrics

You can use the Oracle Application Server built-in performance metrics to analyze OC4J and J2EE application performance. Before you attempt to monitor OC4J performance, verify that the OC4J instance named home that is installed by default with Oracle Application Server is running by accessing the following URL:

```
http://myhost:port/j2ee/
```

The value for *myhost* should be the host where OC4J is installed. The *port* must be the port number on which Oracle HTTP Server listens, as configured in the Oracle HTTP Server `httpd.conf` file.

Be sure to include the trailing slash (`/`) in the URL, otherwise the page cannot be found on the system. If your default Web site has been mapped to something other than the default location `/j2ee/`, then you should access the location configured on your system.

If the default OC4J instance is running, then accessing this URL displays the Welcome page for Oracle Application Server Containers for J2EE (OC4J). From the OC4J Welcome page you can access the samples for JSPs and servlets. If you do not have active J2EE applications that you want to monitor, you can test the monitoring facilities using your browser to request sample servlet-generated or JSP-generated Web pages.

For example, use the following URLs:

```
http://myhost:myport/j2ee/servlet/SnoopServlet
http://myhost:myport/j2ee/servlet/HelloWorldServlet
```

Then, use `AggreSpy` or `dmstool` to see the values of metrics for the built-in performance metrics.

For example, to use `AggreSpy`, enter the following URL in your Web browser:

```
http://myhost:myport/dms0/AggreSpy
```

The resulting display from the `AggreSpy` provides a list of metric tables in the left-hand pane that can be selected to display performance metrics for OC4J and Oracle Application Server components. Alternatively, you can use `dmstool` on the command line or in scripts that you write to display performance metrics.

Note the following when you are monitoring OC4J built-in metrics:

- Oracle recommends that you monitor usage counts and service times for each of your application's Servlets, JSPs, EJBs, JMS applications, and other components, checking collected metrics against your design and deployment assumptions. You should check these assumptions with single browser client scenarios, with simulated multiuser workloads, and in production.
- When troubleshooting performance degradations, you can use either the `AggreSpy` metric tables or the `dmstool` collected metrics to find the Servlets, JSPs, EJBs, EJB methods, and JMS topics or queues that are used most often. In many cases, heavily-used application components are responsible for system resource utilization, so focus your troubleshooting effort on the most heavily-used components first.
- Select the JVM metric table to analyze overall JVM performance for the applications in an OC4J instance. The JVM metric table provides useful information about threads and heap memory allocation. You should check these values to make sure that JVM resources are utilized within expected ranges.

See Also:

- "Viewing Performance Metrics Using AggreSpy" on page 2-5
- "Viewing Performance Metrics Using dmstool" on page 2-10
- Chapter 6, "Optimizing J2EE Applications In OC4J"
- Appendix A, "Performance Metrics" for descriptions of the built-in performance metrics

5

Optimizing Oracle HTTP Server

This chapter discusses the techniques for optimizing Oracle HTTP Server performance in Oracle Application Server.

This chapter contains:

- TCP Tuning Parameters (for UNIX)
- Network Tuning (for Windows)
- Configuring Oracle HTTP Server Directives
- Oracle HTTP Server Logging Options
- Oracle HTTP Server Security Performance Considerations
- Oracle HTTP Server Performance Tips
- Setting mod_oc4j Load Balancing Policies

TCP Tuning Parameters (for UNIX)

Correctly tuned TCP parameters can improve performance dramatically. This section contains recommendations for TCP tuning and a brief explanation of each parameter.

Table 5–1 contains recommended TCP parameter settings and includes references to discussions of each parameter.

Table 5–1 TCP Parameter Settings for Solaris Operating System (SPARC)

Parameter	Setting	Comments
tcp_conn_hash_size	32768	See "Increasing TCP Connection Table Access Speed" on page 5-6
tcp_conn_req_max_q	1024	See "Increasing the Handshake Queue Length" on page 5-8
tcp_conn_req_max_q0	1024	See "Increasing the Handshake Queue Length" on page 5-8
tcp_recv_hiwat	32768	See "Changing the Data Transfer Window Size" on page 5-9
tcp_slow_start_initial	2	See "Changing the Data Transmission Rate" on page 5-8
tcp_time_wait_interval	60000	See "Specifying Retention Time for Connection Table Entries" on page 5-7
tcp_xmit_hiwat	32768	See "Changing the Data Transfer Window Size" on page 5-9

Table 5–2 TCP Parameter Settings for HP-UX

Parameter	Scope	Default Value	Tuned Value	Comments
tcp_time_wait_interval	ndd/dev/tcp	60,000	60,000	See "Specifying Retention Time for Connection Table Entries" on page 5-7
tcp_conn_req_max	ndd/dev/tcp	20	1,024	See "Increasing the Handshake Queue Length" on page 5-8
tcp_ip_abort_interval	ndd/dev/tcp	600,000	60,000	
tcp_keepalive_interval	ndd/dev/tcp	7,20,00,000	900,000	
tcp_rexmit_interval_initial	ndd/dev/tcp	1,500	1,500	
tcp_rexmit_interval_max	ndd/dev/tcp	60,000	60,000	

Table 5–2 TCP Parameter Settings for HP-UX

Parameter	Scope	Default Value	Tuned Value	Comments
tcp_rexmit_interval_min	ndd/dev/tcp	500	500	
tcp_xmit_hiwater_def	ndd/dev/tcp	32,768	32,768	See "Changing the Data Transfer Window Size" on page 5-9
tcp_recv_hiwater_def	ndd/dev/tcp	32,768	32,768	See "Changing the Data Transfer Window Size" on page 5-9

Table 5–3 TCP Parameter Settings for Tru64

Parameter	Module	Default value	Tuned Value	Comments
tcbhashsize	sysconfig -r inet	512	16,384	See "Increasing TCP Connection Table Access Speed" on page 5-6
tcbhashnum	sysconfig -r inet	1	16 (as of 5.0)	
tcp_keepalive_default	sysconfig -r inet	0	1	
tcp_sendspace	sysconfig -r inet	16,384	65,535	
tcp_recvspace	sysconfig -r inet	16,384	65,535	
somaxconn	sysconfig -r socket	1,024	65,535	
sominconn	sysconfig -r socket	0	65,535	
sbcompress_threshold	sysconfig -r socket	0	600	

Table 5–4 TCP Parameter Settings for AIX

Parameter	Model	Default Value	Recommended Value	Comments
rfc1323	/etc/rc.net	0	1	
sb_max	/etc/rc.net	65,536	1,31,072	
tcp_mssdflt	/etc/rc.net	512	1,024	
ipqmaxlen	/etc/rc.net	50	100	

Table 5–4 TCP Parameter Settings for AIX

Parameter	Model	Default Value	Recommended Value	Comments
tcp_sendspace	/etc/rc.net	16,384	65,536	
tcp_recvspace	/etc/rc.net	16,384	65,536	
xmt_que_size	/etc/rc.net	30	150	

Tuning Linux

Raising Network Limits on Linux Systems for 2.1.100 or greater

Linux only allows you to use 15 bits of the TCP window field. This means that you have to multiply everything by 2, or recompile the kernel without this limitation.

See Also: "Tuning at Compile Time" on page 5-5

Tuning a Running System

There is no `sysctl` application for changing kernel values. You can change the kernel values with an editor such as `vi`.

Tuning the Default and Maximum Size

Edit the following files to change kernel values.

Table 5–5 Linux TCP Parameters

Filename	Details
/proc/sys/net/core/rmem_default	Default Receive Window
/proc/sys/net/core/rmem_max	Maximum Receive Window
/proc/sys/net/core/wmem_default	Default Send Window
/proc/sys/net/core/wmem_max	Maximum Send Window

You will find some other possibilities to tune TCP in `/proc/sys/net/ipv4/`:

- tcp_timestamps
- tcp_windowscaling
- tcp_sack

There is a brief description of TCP parameters in `/Documentation/networking/ip-sysctl.txt.`

Tuning at Compile Time

All the preceding TCP parameter values are set by default by a header file in the Linux kernel source directory `/LINUX-SOURCE-DIR/include/linux/skbuff.h`

These values are the defaults. This is run time configurable.

```
# ifdef CONFIG_SKB_LARGE
#define SK_WMEM_MAX 65535
#define SK_RMEM_MAX 65535
# else
#define SK_WMEM_MAX 32767
#define SK_RMEM_MAX 32767
#endif
```

You can change the `MAX-WINDOW` value in the Linux kernel source directory in the file `/LINUX-SOURCE-DIR/include/net/tcp.h.`

```
#define MAX_WINDOW 32767
#define MIN_WINDOW 2048
```

Note: Never assign values greater than 32767 to windows, without using window scaling.

The `MIN_WINDOW` definition limits you to using only 15bits of the window field in the TCP packet header.

For example, if you use a 40kB window, set the `rmem_default` to 40kB. The stack will recognize that the value is less than 64 kB, and will not negotiate a winshift. But due to the second check, you will get only 32 kB. So, you need to set the `rmem_default` value at greater than 64 kB to force a winshift=1. This lets you express the required 40 kB in only 15 bits.

With the tuned TCP stacks, it was possible to get a maximum throughput between 1.5 and 1.8 Mbits through a 2Mbit satellite link, measured with netperf.

Setting TCP Parameters

To set the connection table hash parameter for the Solaris Operating System, you must add the following line to the `/etc/system` file, and then restart the system:

```
set tcp:tcp_conn_hash_size=32768
```

On Tru64, set `tcbhashsize` in the `/etc/sysconfigtab` file.

A sample script, `tcpset.sh`, that changes TCP parameters to the settings recommended here, is included in the `$ORACLE_HOME/Apache/Apache/bin/` directory.

> **Note:** If your system is restarted after you run the script, the default settings will be restored and you will have to run the script again. To make the settings permanent, enter them in your system startup file.

Increasing TCP Connection Table Access Speed

If you have a large user population, you should increase the **hash** size for the TCP connection table. The **hash** size is the number of **hash** buckets used to store the connection data. If the buckets are very full, it takes more time to find a connection. Increasing the **hash** size reduces the connection lookup time, but increases memory consumption.

Suppose your system performs 100 connections per second. If you set `tcp_time_wait_interval` to `60000`, then there will be about `6000` entries in your TCP connection table at any time. Increasing your **hash** size to `2048` or `4096` will improve performance significantly.

On a system servicing 300 connections per second, changing the **hash** size from the default of `256` to a number close to the number of connection table entries decreases the average round trip time by up to three to four seconds. The maximum **hash** size is `262144`. Ensure that you increase memory as needed.

To set the `tcp_conn_hash_size` for the Solaris Operating System, add the following line to the `/etc/system` file. The parameter will take effect when the system is restarted.

```
set tcp:tcp_conn_hash_size=32768
```

On Tru64, set `tcbhashsize` in the `/etc/sysconfigtab` file.

Specifying Retention Time for Connection Table Entries

As described in the previous section, when a connection is established, the data associated with it is maintained in the TCP connection table. On a busy system, much of TCP performance (and by extension web server performance) is governed by the speed with which the entry for a specific TCP connection can be accessed in the connection table. The access speed depends on the number of entries in the table, and on how the table is structured (for example, its hash size). The number of entries in the table depends both on the rate of incoming requests, and on the lifetime of each connection.

For each connection, the server maintains the TCP connection table entry for some period after the connection is closed so it can identify and properly dispose of any leftover incoming packets from the client. The length of time that a TCP connection table entry will be maintained after the connection is closed can be controlled with the `tcp_time_wait_interval` parameter. The default for the Solaris Operating System for this parameter is 240,000 ms in accordance with the TCP standard. The four minute setting on this parameter is intended to prevent congestion on the Internet due to error packets being sent in response to packets which should be ignored. In practice, 60,000 ms is sufficient, and is considered acceptable. This setting will greatly reduce the number of entries in the TCP connection table while keeping the connection long enough to discard most, if not all, leftover packets associated with it. We therefore suggest you set:

On HP-UX and for Solaris Operating System 2.7 and higher:

```
/usr/sbin/ndd -set /dev/tcp tcp_time_wait_interval 60000
```

Note: If your user population is widely dispersed with respect to Internet topology, you may want to set this parameter to a higher value. You can improve access time to the TCP connection table with the `tcp_conn_hash_size` parameter.

Increasing the Handshake Queue Length

During the TCP connection handshake, the server, after receiving a request from a client, sends a reply, and waits to hear back from the client. The client responds to the server's message and the handshake is complete. Upon receiving the first request from the client, the server makes an entry in the listen queue. After the client responds to the server's message, it is moved to the queue for messages with completed handshakes. This is where it will wait until the server has resources to service it.

The maximum length of the queue for incomplete handshakes is governed by `tcp_conn_req_max_q0`, which by default is `1024`. The maximum length of the queue for requests with completed handshakes is defined by `tcp_conn_req_max_q`, which by default is `128`.

On most web servers, the defaults will be sufficient, but if you have several hundred concurrent users, these settings may be too low. In that case, connections will be dropped in the handshake state because the queues are full. You can determine whether this is a problem on your system by inspecting the values for `tcpListenDrop`, `tcpListenDropQ0`, and `tcpHalfOpenDrop` with `netstat -s`. If either of the first two values are nonzero, you should increase the maximums.

The defaults are probably sufficient, but Oracle recommends that you increase the value of `tcp_conn_req_max_q` to `1024`. You can set these parameters with:

On the Solaris Operating System:

```
% /usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 1024
% /usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q0 1024
```

On HP-UX:

```
prompt>/usr/sbin/ndd-set /dev/tcp tcp_conn_req_max 1024
```

Changing the Data Transmission Rate

TCP implements a slow start data transfer to prevent overloading a busy segment of the Internet. With slow start, one packet is sent, an acknowledgment is received, then two packets are sent. The number sent to the server continues to be doubled after each acknowledgment, until the TCP transfer window limits are reached.

Unfortunately, some operating systems do not immediately acknowledge the receipt of a single packet during connection initiation. By default, the Solaris Operating System sends only one packet during connection initiation, per the TCP standard. This can increase the connection startup time significantly. We therefore recommend increasing the number of initial packets to two when initiating a data transfer. This can be accomplished using the following command:

```
% /usr/sbin/ndd -set /dev/tcp tcp_slow_start_initial 2
```

Changing the Data Transfer Window Size

The size of the TCP transfer windows for sending and receiving data determine how much data can be sent without waiting for an acknowledgment. The default window size is 8192 bytes. Unless your system is memory constrained, these windows should be increased to the maximum size of 32768. This can speed up large data transfers significantly. Use these commands to enlarge the window:

On Solaris Operating System:

```
% /usr/sbin/ndd -set /dev/tcp tcp_xmit_hiwat 32768
% /usr/sbin/ndd -set /dev/tcp tcp_recv_hiwat 32768
```

On HP-UX:

```
prompt>/usr/sbin/ndd -set /dev/tcp tcp_xmit_hiwater_def 32768
prompt>/usr/sbin/ndd -set /dev/tcp tcp_recv_hiwater_def 32768
```

Because the client typically receives the bulk of the data, it would help to enlarge the TCP receive windows on end users' systems, as well.

Network Tuning (for Windows)

To maximize network performance on your system (after ensuring that memory is sufficient) you should:

- Run only the TCP/IP protocol on your system
- Use the Maximize Throughput for File Sharing (Server Properties) setting for TCP optimization

To access these, select Settings from the Start menu, then go to the Control panel and select Network. If you make changes to the settings, you must restart the system.

Setting the TCP Protocol

To set the TCP Protocol:

1. In the Network dialog box, select the Protocols tab.
2. Remove protocols other than TCP/IP from the list box.
3. Restart the system.

Setting Server Properties

To set Server Properties:

1. In the Network dialog box, select the Services tab.
2. Select Server, and click Properties.
3. Ensure that the Maximize Throughput for File Sharing setting is in effect.
4. Restart the system.

In-house tests have shown that when either Maximize Throughput for File Sharing or Maximize Throughput for Network Applications is in effect, performance is much better than with any other setting. Comparing response times under load, we found that the Maximize Throughput for File Sharing setting is twice as fast as Maximize Throughput for Network Applications.

Caution: Oracle recommends against tuning individual TCP/IP parameters in the registry without devoting significant time to thoroughly test the impact of each parameter change on your environment.

There are a number of things to keep in mind when running Oracle HTTP Server.

1. Be certain that you have sufficient memory. System memory usage can be monitored by watching the display under the performance tab in the Task Manager.
2. Be certain that only the TPC/IP protocol stack is running. If another protocol is running, it will be listed in the list under the Protocols tab of the Control Panel/Network dialog box. To remove it, select it with the mouse, and click Remove. If you close the Network dialog box, you will be prompted to restart the system. It will be easier, however, to first continue with step 3.

3. Select the "Maximize Throughput for File Sharing" network optimization scheme. Under the Services tab of the Control Panel / Network dialog box, you can examine the Server properties. Select "Server" in the list, and click Properties. This will bring up a dialog box that allows you to choose the criteria for which TCP will be optimized. The default setting is "Maximize Throughput for File Sharing". We recommend you use the setting. If this has been otherwise set, reset it to the default and click "OK". Then close the Control Panel / Network dialog box. If you changed this setting, you will be prompted to restart the system, and if you have made any changes in the preceding Step 1, or in this box, you should do so.

> **Note:** The performance is much better when either "Maximize Throughput for File Sharing" or "Maximize Throughput for Network Applications" is chosen, than when either of the other options is chosen. We have also seen that the response time under load is cut in half when we maximize for file sharing rather than for network applications.

In addition to the preceding steps, one can adjust individual TCP / IP parameters in the registry. We do not recommend that you do so as it is complex. Unless you have plenty of time to test the impact for your environment, we recommend you limit your TCP / IP tuning to the preceding steps.

Configuring Oracle HTTP Server Directives

Oracle HTTP Server uses directives in `httpd.conf` to configure the application server. This configuration file specifies the maximum number of HTTP requests that can be processed simultaneously, logging details, and certain timeouts.

Table 5–6 lists directives that may be significant for performance.

Table 5–6 Oracle HTTP Server Configuration Properties

Directive	Description
`ListenBackLog`	Specifies the maximum length of the queue of pending connections. Generally no tuning is needed or desired. Note that some Operating Systems do not use exactly what is specified as the backlog, but use a number based on, but normally larger than, what is set. Default Value: 511
`MaxClients`	Specifies a limit on the total number of servers running, that is, a limit on the number of clients who can simultaneously connect. If the number of client connections reaches this limit, then subsequent requests are queued in the TCP/IP system up to the limit specified with the `ListenBackLog` directive (after the queue of pending connections is full, new requests generate connection errors until a process becomes available). The maximum allowed value for `MaxClients` is 8192 (8K). Default Value: 150
`MaxRequestsPerChild`	The number of requests each child process is allowed to process before the child dies. The child will exit so as to avoid problems after prolonged use when Apache (and maybe the libraries it uses) leak memory or other resources. On most systems, this isn't really needed, but some UNIX systems have notable leaks in the libraries. For these platforms, set `MaxRequestsPerChild` to something like 10000 or so; a setting of 0 means unlimited. This value does not include `KeepAlive` requests after the initial request per connection. For example, if a child process handles an initial request and 10 subsequent "keptalive" requests, it would only count as 1 request toward this limit. **Note**: On Windows systems `MaxRequestsPerChild` should always be set to `0` (unlimited). On Windows there is only one server process, so it is not a good idea to limit this process.

`MaxSpareServers`
 	
`MinSpareServers`
	Server-pool size regulation. Rather than making you guess how many server processes you need, Oracle HTTP Server dynamically adapts to the load it sees, that is, it tries to maintain enough server processes to handle the current load, plus a few spare servers to handle transient load spikes (for example, multiple simultaneous requests from a single Netscape browser).
	
	It does this by periodically checking how many servers are waiting for a request. If there are fewer than `MinSpareServers`, it creates a new spare. If there are more than `MaxSpareServers`, some of the spares die off.
	
	The default values are probably ok for most sites.
	
	Default Values:
	
	`MaxSpareServers`: 10
	
	`MinSpareServers`: 5

Table 5–6 (Cont.) Oracle HTTP Server Configuration Properties

Directive	Description
`StartServers`	Number of servers to start initially. If you expect a sudden load after restart, set this value based on the number child servers required. Default Value: 5
`Timeout`	The number of seconds before incoming receives and outgoing sends time out. Default Value: 300
`KeepAlive`	Whether or not to allow persistent connections (more than one request per connection). Set to `Off` to deactivate. Default Value: On
`MaxKeepAliveRequests`	The maximum number of requests to allow during a persistent connection. Set to 0 to allow an unlimited amount. If you have long client sessions, you might want to increase this value. Default Value: 100
`KeepAliveTimeout`	Number of seconds to wait for the next request from the same client on the same connection. Default Value: 15 seconds

Configuring the MaxClients Directive

The `MaxClients` directive limits the number of clients that can simultaneously connect to your web server, and thus the number of httpd processes. You can configure this parameter in the `httpd.conf` file up to a maximum of 8K (the default value is 150).

Tests on a previous release, with static page requests (average size 20K) on a 2 processor, system showed that:

- The default `MaxClients` setting of 150 was sufficient to saturate the network.
- Approximately 60 httpd processes were required to support 300 concurrent users (no think time).

On the system described, and on 4 and 6-processor systems, there was no significant performance improvement in increasing the `MaxClients` setting from 150 to 256, based on static page and servlet tests with up to 1000 users.

Increasing `MaxClients` when system resources are saturated does not improve performance. When there are no httpd processes available, connection requests are queued in the TCP/IP system until a process becomes available, and eventually clients terminate connections. If you are using persistent connections, you may require more concurrent httpd server processes.

For dynamic requests, if the system is heavily loaded, it might be better to allow the requests to queue in the network (thereby keeping the load on the system manageable). The question for the system administrator is whether a timeout error and retry is better than a long response time. In this case, the `MaxClients` setting could be reduced, as a throttle on the number of concurrent requests on the server.

The `MaxClients` parameter on UNIX systems works like the `ThreadsPerChild` parameter on Windows systems.

See Also: "Configuring the ThreadsPerChild Parameter (for Windows)" on page 5-14

How Persistent Connections Can Reduce httpd Process Availability

The default settings for the `KeepAlive` directives are:

```
KeepAlive on
MaxKeepAliveRequests 100
KeepAliveTimeOut 15
```

These settings allow enough requests per connection and time between requests to reap the benefits of the persistent connections, while minimizing the drawbacks. You should consider the size and behavior of your own user population in setting these values on your system. For example, if you have a large user population and the users make small infrequent requests, you may want to reduce the above settings, or even set `KeepAlive` to off. If you have a small population of users that return to your site frequently, you may want to increase the settings.

Configuring the ThreadsPerChild Parameter (for Windows)

The `ThreadsPerChild` parameter in the `httpd.conf` file specifies the number of requests that can be handled concurrently by the HTTP server. Requests in excess of the `ThreadsPerChild` parameter value wait in the TCP/IP queue. Allowing the requests to wait in the TCP/IP queue often results in the best **response time** and **throughput**.

The `ThreadsPerChild` parameter on Windows systems works like the `MaxClients` parameter on UNIX systems.

See Also: "Configuring the MaxClients Directive" on page 5-13

Configuring ThreadsPerChild for Static Page Requests

The more concurrent threads you make available to handle requests, the more requests your server can process. But be aware that with too many threads, under high load, requests will be handled more slowly and the server will consume more system resources.

In in-house tests of static page requests, a setting of 20 `ThreadsPerChild` per CPU produced good **response time** and **throughput** results. For example, if you have four CPUs, set `ThreadsPerChild` to 80. If, with this setting, CPU utilization does not exceed 85%, you can increase `ThreadsPerChild`, but ensure that the available threads are in use.

Oracle HTTP Server Logging Options

This section discusses types of logging, log levels, and the performance implications for using logging.

Access Logging

For static page requests, access logging of the default fields results in a 2-3% performance cost.

Configuring the HostNameLookups Directive

By default, the `HostNameLookups` directive is set to `Off`. The server writes the IP addresses of incoming requests to the log files. When `HostNameLookups` is set to on, the server queries the DNS system on the Internet to find the host name associated with the IP address of each request, then writes the host names to the log.

Performance degraded by about 3% (best case) in Oracle in-house tests with `HostNameLookups` set to on. Depending on the server load and the network connectivity to your DNS server, the performance cost of the DNS lookup could be high. Unless you really need to have host names in your logs in real time, it is best to log IP addresses.

On UNIX systems, you can resolve IP addresses to host names off-line, with the `logresolve` utility found in the `$ORACLE_HOME/Apache/Apache/bin/` directory.

Error logging

The server notes unusual activity in an error log. The ErrorLog and LogLevel directives identify the log file and the level of detail of the messages recorded. The default level is warn. There was no difference in static page performance on a loaded system between the warn, info, and debug levels.

For requests that use dynamic resources, for example requests that use mod_osso, mod_plsql, or mod_oc4j, there is a performance cost associated with setting higher debugging levels, such as the debug level.

Oracle HTTP Server Security Performance Considerations

This section covers the following topics:

- Oracle HTTP Server Secure Sockets Layer (SSL) Performance Issues
- Oracle HTTP Server Port Tunneling Performance Issues

Oracle HTTP Server Secure Sockets Layer (SSL) Performance Issues

Secure Sockets Layer (SSL) is a protocol developed by Netscape Communications Corporation that provides authentication and encrypted communication over the Internet. Conceptually, SSL resides between the application layer and the transport layer on the protocol stack. While SSL is technically an application-independent protocol, it has become a standard for providing security over HTTP, and all major web browsers support SSL.

SSL can become a bottleneck in both the responsiveness and the scalability of a web-based application. Where SSL is required, the performance challenges of the protocol should be carefully considered. Session management, in particular session creation and initialization, is generally the most costly part of using the SSL protocol, in terms of performance.

This section covers the following SSL Performance related information:

- Oracle HTTP Server SSL Caching
- SSL Application Level Data Encryption
- SSL Performance Recommendations

See Also: *Oracle Application Server 10g Security Guide*

Oracle HTTP Server SSL Caching

When an SSL connection is initialized, a session based handshake between client and server occurs that involves the negotiation of a cipher suite, the exchange of a private key for data encryption, and server and, optionally, client authentication through digitally-signed certificates.

After the SSL session state has been initiated between a client and a server, the server can avoid the session creation handshake in subsequent SSL requests by saving and reusing the session state. The Oracle HTTP Server caches a client's Secure Sockets Layer (SSL) session information by default. With session caching, only the first connection to the server incurs high latency.

The `SSLSessionCacheTimeout` directive in `httpd.conf` determines how long the server keeps a saved SSL session (the default is `300` seconds). Session state is discarded if it is not used after the specified time period, and any subsequent SSL request must establish a new SSL session and begin the handshake again. The `SSLSessionCache` directive specifies the location for saved SSL session information, the default location on UNIX is the `$ORACLE_HOME/Apache/Apache/logs/` directory or on Windows systems, `%ORACLE_HOME%\Apache\Apache\logs\`. Multiple Oracle HTTP Server processes can use a saved session cache file.

Saving SSL session state can significantly improve performance for applications using SSL. For example, in a simple test to connect and disconnect to an SSL-enabled server, the elapsed time for 5 connections was 11.4 seconds without SSL session caching. With SSL session caching enabled, the elapsed time for 5 round trips was 1.9 seconds.

The reuse of saved SSL session state has some performance costs. When SSL session state is stored to disk, reuse of the saved state normally requires locating and retrieving the relevant state from disk. This cost can be reduced when using HTTP persistent connections. Oracle HTTP Server uses persistent HTTP connections by default, assuming they are supported on the client side. In HTTP over SSL as implemented by Oracle HTTP Server, SSL session state is kept in memory while the associated HTTP connection is persisted, a process which essentially eliminates the overhead of SSL session reuse (conceptually, the SSL connection is kept open along with the HTTP connection).

SSL Application Level Data Encryption

In most applications using SSL, the data encryption cost is small compared with the cost of SSL session management. Encryption costs can be significant where the volume of encrypted data is large, and in such cases the data encryption algorithm and key size chosen for an SSL session can be significant.

In general there is a trade-off between security level and performance. For example, on a modern processor, RSA estimates its RC4 cipher to take in the vicinity of 8-16 machine operations per output byte. Standard DES encryption will incur roughly 8 times the overhead of RC4, and triple DES will take about 25 times the overhead of DES. However, when using triple DES, the encryption costs will not be noticeable in most applications. Oracle HTTP Server supports these three cipher suites, and other cipher suites as well.

Oracle HTTP Server negotiates a cipher suite with a client based on the `SSLCipherSuite` attribute specified in `httpd.conf`.

> **See Also:** *Oracle HTTP Server Administrator's Guide* for information on using supported cipher suites

SSL Performance Recommendations

The following recommendations can assist you with determining performance requirements when working with Oracle HTTP Server and SSL.

1. The SSL handshake is an inherently expensive process in terms of both CPU usage and response time. Thus, use SSL only where needed. Determine the parts of the application that require the security, and the level of security required, and protect only those parts at the requisite security level. Attempt to minimize the need for the SSL handshake by using SSL sparingly, and by reusing session state as much as possible. For example, if a page contains a small amount of sensitive data and a number of non-sensitive graphic images, use SSL to transfer the sensitive data only, use normal HTTP to transfer the images. If the application requires server authentication only, do not use client authentication. If the performance goals of an application cannot be met by this method alone, additional hardware may be required.
2. Design the application to use SSL efficiently. Group secure operations together to take advantage of SSL session reuse and SSL connection reuse.
3. Use persistent connections, if possible, to minimize cost of SSL session reuse.
4. Tune the session cache timeout value (the `SSLSessionCacheTimeout` attribute in `httpd.conf`). A trade-off exists between the cost of maintaining an SSL session cache and the cost of establishing a new SSL session. As a rule, any secured business process, or conceptual grouping of SSL exchanges, should be completed without incurring session creation more than once. The default value

for the `SSLSessionCacheTimeout` attribute is 300 seconds. It is a good idea to test an application's usability to help tune this setting.

5. If large volumes of data are being protected through SSL, pay close attention to the cipher suite being used. The `SSLCipherSuite` directive specified in `httpd.conf` controls the cipher suite. If lower levels of security are acceptable, use a less-secure protocol using a smaller key size (this may improve performance significantly). Finally, test the application using each available cipher suite for the desired security level to find the most performant suite.
6. Having taken the preceding considerations into account, if SSL remains a bottleneck to the performance and scalability of your application, consider deploying multiple Oracle HTTP Server instances over a hardware cluster or consider the use of SSL accelerator cards.

Oracle HTTP Server Port Tunneling Performance Issues

When OracleAS Port Tunneling is configured, every request processed passes through the OracleAS Port Tunneling infrastructure. Thus, using OracleAS Port Tunneling can have an impact on the overall Oracle HTTP Server request handling performance and scalability.

With the exception of the number of OracleAS Port Tunneling processes to run, the performance of OracleAS Port Tunneling is self tuning. The only performance control available is to start more OracleAS Port Tunneling processes, this increases the number of available connections and hence the scalability of the system.

The number of OracleAS Port Tunneling processes is based on the degree of availability required, and the number of anticipated connections. This number can not be automatically determined because for each additional process a new port must be opened through the firewall between the DMZ and the intranet. You cannot start more processes than you have open ports, and you do not want less processes than open ports, since in this case ports would not have any process bound to them.

To measure the iasPT performance, determine the request time for servlet requests that pass through the OracleAS Port Tunneling infrastructure. The response time of an Oracle Application Server instance running with OracleAS Port Tunneling should be compared with a system without OracleAS Port Tunneling to determine whether your performance requirements can be met using OracleAS Port Tunneling.

> **See Also:** *Oracle HTTP Server Administrator's Guide* for information on configuring OracleAS Port Tunneling

Oracle HTTP Server Performance Tips

The following tips can enable you to avoid or debug potential Oracle HTTP Server (OHS) performance problems:

- Analyze Static Versus Dynamic Requests
- Analyze Time Differences Between Oracle HTTP Server and OC4J Servers
- Beware of a Single Data Point Yielding Misleading Results

Analyze Static Versus Dynamic Requests

It is important to understand where your server is spending resources so you can focus your tuning efforts in the areas where the most stands to be gained. In configuring your system, it can be useful to know what percentage of the incoming requests are static and what percentage are dynamic.

Static pages can be cached by Oracle Application Server Web Cache, if it is in use. Generally, you want to concentrate your tuning effort on dynamic pages because dynamic pages can be costly to generate. Also, by monitoring and tuning your application, you may find that much of the dynamically generated content, such as catalog data, can be cached, sparing significant resource usage.

See Also:

- Chapter 3, "Monitoring Oracle HTTP Server"
- Chapter 7, "Optimizing OracleAS Web Cache"

Analyze Time Differences Between Oracle HTTP Server and OC4J Servers

In some cases, you may notice a high discrepancy between the average time to process a request in Oracle Application Server Containers for J2EE (OC4J) and the average response time experienced by the user. If the time is not being spent actually doing the work in OC4J, then it is probably being spent in transport.

If you notice a large discrepancy between the request processing time in OC4J and the average response time, consider tuning the Oracle HTTP Server directives shown in the section, "Configuring Oracle HTTP Server Directives" on page 5-11.

Beware of a Single Data Point Yielding Misleading Results

You can get unrepresentative results when data outliers appear. This can sometimes occur at start-up. To simulate a simple example, assume that you ran a PL/SQL "Hello, World" application for about 30 seconds. Examining the results, you can see that the work was all done in `mod_plsql.c`:

```
/ohs_server/ohs_module/mod_plsql.c
  handle.maxTime:     859330
  handle.minTime:      17099
  handle.avg:          19531
  handle.active:           0
  handle.time:      24023499
  handle.completed:     1230
```

Note that `handle.maxTime` is much higher than `handle.avg` for this module. This is probably because when the first request is received, a database connection must be opened. Later requests can make use of the established connection. In this case, to obtain a better estimate of the average service time for a PL/SQL module, that does not include the database connection open time which causes the `handle.maxTime` to be very large, recalculate the average as in the following:

```
(time - maxTime)/(completed -1)
```

For example, in this case this would be:

```
(24023499 - 859330)/(1230 -1) = 18847.98
```

Setting mod_oc4j Load Balancing Policies

At many sites Oracle Application Server uses the Oracle HTTP Server module `mod_oc4j` to load balance incoming stateless HTTP requests. By selecting the appropriate load balancing policy for `mod_oc4j` you can improve performance on your site.

The `mod_oc4j` module supports several configurable load balancing policies, including the following:

- Round robin routing (this is the default `mod_oc4j` load balancing policy)
- Random routing
- Round robin or random with local affinity routing, using the `local` option
- Round robin or random with host-level weighted routing, using the `weighted` option

---

Note: For a session based request `mod_oc4j` always directs the request to the original OC4J process which created the session, unless the original OC4J process is not available. In case of failure, `mod_oc4j` sends the request to another OC4J process with the same island name as the original request (either within same host if available, or on a remote host).

This section covers the following topics:

- Quick Summary for Using Load Balancing With mod_oc4j
- Using Round Robin and Random Policies With mod_oc4j Load Balancing
- Using Local Affinity Option With mod_oc4j Load Balancing
- Using Weighted Routing Option With mod_oc4j Load Balancing
- Recommendations for Load Balancing With mod_oc4j

Quick Summary for Using Load Balancing With mod_oc4j

This section provides a quick summary of the load balancing configuration you may want to use when configuring mod_oc4j for Oracle Application Server:

- When Oracle Application Server runs in a single host with one or more OC4J Instances, we recommend using either the round robin or random load balancing policy. The performance characteristics for the particular policy can depend on the applications that run on your site; however, in many cases these two policies will yield similar performance.
- When Oracle Application Server is configured at a site that uses multiple hosts with the same hardware and Oracle Application Server configurations, we recommend using either round robin with the local affinity option or random with the local affinity option.
- When Oracle Application Server is configured at a site that uses multiple hosts with different hardware and different Oracle Application Server configurations, we recommend using either round robin with the weighted option or random with the weighted option. For sites where it is difficult to determine how much load each host can handle, and it is difficult to assign an accurate routing weight, you may want to use either round robin with the local affinity option or random with the local affinity option.

See Also: *Oracle HTTP Server Administrator's Guide* for a description of `mod_oc4j` configuration options

Using Round Robin and Random Policies With mod_oc4j Load Balancing

Using round robin routing or random routing, without the `local` or `weighted` options, specifies that `mod_oc4j` creates a list of all the available OC4J processes across all hosts. For incoming requests, `mod_oc4j` routes the requests using the list of available OC4J processes, either selecting processes from the list randomly, or using a round robin selection policy (with the round robin, the first request is selected randomly, and requests after that are selected using the round robin policy.

If you use either of these load balancing policies, you need to consider the number of OC4J processes that you run on each host. Without specifying the weighted routing option for `mod_oc4j`, if you configure your site to start different numbers of OC4J processes on each host, this causes an implicit weighting to occur where more requests are sent to hosts with more OC4J processes. If this implicit weighting of requests by the number of OC4J processes per host is not what you want, then you should consider specifying a routing weight for each host and using the weighted option.

Note: In many cases the round robin and random policies will yield similar performance.

For example, if you use the default round robin load balancing policy and you start 4 OC4J processes on `Host_A` and 1 OC4J process on `Host_B`, then `mod_oc4j` sends 4 requests to `Host_A` for each 1 request that it sends to `Host_B`. Thus, with this configuration you are implicitly sending 4 times as many requests to `Host_A`.

See Also: "Using Weighted Routing Option With mod_oc4j Load Balancing" on page 5-24

Using Local Affinity Option With mod_oc4j Load Balancing

Selecting the local affinity option tells `mod_oc4j` to always try to select the local OC4J instance to service incoming requests. When no local OC4J processes are available, `mod_oc4j` selects from a list of available remote OC4J processes. You can select either the round robin or the random policies with the local affinity option.

For example to select the round robin policy with local affinity, specify the following directive in `mod_oc4j.conf`:

```
Oc4jSelectMethod roundrobin:local
```

Using Weighted Routing Option With mod_oc4j Load Balancing

Selecting the weighted routing option specifies that `mod_oc4j` should distribute HTTP requests across the available hosts and use a specified routing weight to calculate the distribution of incoming requests that are sent to each host. The routing weight is specified with the `Oc4jRoutingWeight` directive. You can specify either the round robin or the random policies with the weighted option.

For example, if the routing weight set for `Host_A` is 3 and the routing weight set for `Host_B` is 1, this specifies that `Host_A` should be sent three times the number of requests as compared to `Host_B`.

Note: Using weighted routing, incoming requests are routed according the specified routing weight and without consideration for the number of OC4J processes running on each host.

To configure the mod_oc4j module in Oracle HTTP Server to specify round robin with a routing weight of 3 for Host_A and a routing weight of 1 for Host_B, add the following directives to mod_oc4j.conf:

```
Oc4jSelectMethod   roundrobin:weighted
Oc4jRoutingWeight  Host_A 3
```

In this example you do not need to specify a routing weight for Host_B, since the default routing weight is 1.

You need to determine the routing weight for each system based on what other components are running on the systems and based on how many requests each system can adequately handle.

> **Note:** An inaccurate specification for the routing weight could have negative performance implications for your site.

Recommendations for Load Balancing With mod_oc4j

In general, when configuring the mod_oc4j load balancing policy, we recommend the following:

1. If you have multiple systems with similar hardware configuration use round robin with local affinity or random load balancing policy with local affinity.

   For example, if you have multiple hosts with the same number of CPUs with same speed, and the same memory, with Oracle HTTP Server running with the same number of OC4J processes on each host, and you are using a hardware load balancer or web cache in the front end to route the requests to Oracle HTTP Server on each host, then, using either round robin with local affinity or random with local affinity is recommended.

2. If you have multiple systems, each with a similar hardware configuration, and you want to run Oracle HTTP Server only on one host, then select either the round robin with the weighted option or random with the weighted option.

   For example, consider a site with 2 hosts, Host_A and Host_B, each with 2 CPUs. On this site you only run Oracle HTTP Server on Host_A, and each host includes one OC4J instance with one OC4J process. With this configuration, selecting round robin with the weighted option or random with the weighted option, and using a higher routing weight on Host_B will help to shift more requests to Host_B. Since Host_B is not running Oracle HTTP Server this configuration should provide better performance for this site.

3. If you are running Oracle HTTP Server on a separate system which routes the HTTP web requests to multiple hosts running only OC4J and the systems use similar hardware with the same number of OC4J processes, then use round robin or random load balancing policy.
4. If you are running Oracle HTTP Server, OC4J and other Oracle Application Server components on multiple systems which have different hardware configurations, use round robin with the weighted option or random with the weighted option to help distribute requests to each system.

   You need to determine the routing weight for each system based on what other components are running on the systems and based on how many requests each system can adequately handle.

6

Optimizing J2EE Applications In OC4J

This chapter provides guidelines for improving the performance of Oracle Application Server Containers for J2EE (OC4J) applications in Oracle Application Server.

This chapter contains:

- OC4J J2EE Application Performance Quickstart
- Improving J2EE Application Performance by Configuring OC4J Instance
- Setting Java Command Line Options (Using JVM and OC4J Performance Options)
- Setting Up Data Sources – Performance Issues
- Setting server.xml Configuration Parameters
- Improving Servlet Performance in Oracle Application Server
- Improving JSP Performance in Oracle Application Server
- Improving EJB Performance in Oracle Application Server
- Improving Web Services Performance in Oracle Application Server
- Improving BC4J Performance in Oracle Application Server
- Improving JAAS (JAZN) Performance in Oracle Application Server
- Using Multiple OC4Js, Limiting Connections and Load Balancing
- Performance Considerations for Deploying J2EE Applications

Note: This chapter describes using Oracle Enterprise Manager Application Server Control for setting OC4J and application configuration options. You can also use the Distributed Configuration Management (DCM) utility, `dcmctl`, to set configuration options. This utility provides a command-line alternative to using Oracle Enterprise Manager Application Server Control for some Oracle Application Server configuration and management tasks.

OC4J J2EE Application Performance Quickstart

This section provides a quickstart for tuning J2EE applications that run on OC4J, providing links for information on important performance issues.

Table 6–1 lists a quick guide for performance issues for J2EE applications.

Table 6–1 Critical Performance Areas for J2EE Applications

Performance Area	Description and Reference
Providing Adequate Memory Resources	To improve the performance of your J2EE applications, provide adequate memory resources. If the OC4J running your J2EE applications does not have enough memory, performance can suffer due to the overhead required to manage limited memory See "Setting the JVM Heap Size for OC4J Processes" on page 6-4
Caching and Reusing Database Connections	Setting up database connection pooling properly is often a critical performance consideration for J2EE applications that access a database. Data sources provide configuration options that allow you to use and configure pooled database connections. See "Setting Up Data Sources – Performance Issues" on page 6-13
Managing Concurrency and Limiting Connections	See "Limiting Connections" on page 6-75
Load Balancing	See "Configuring Multiple OC4J Processes" on page 6-70
Balancing Applications	See "Load Balancing Applications" on page 6-72
Database Monitoring and Tuning	See Chapter 10, "Database Tuning Considerations" on page 10-1

Improving J2EE Application Performance by Configuring OC4J Instance

Tuning OC4J configuration options lets you improve the performance of J2EE applications running on an OC4J Instance. Modifying the configuration may require balancing the available resources on your system with the performance requirements for your applications.

This section covers configuration changes that can affect J2EE application performance and includes the following topics:

- Setting Java Command Line Options (Using JVM and OC4J Performance Options)
- Setting Up Data Sources – Performance Issues
- Setting server.xml Configuration Parameters

Setting Java Command Line Options (Using JVM and OC4J Performance Options)

Depending on your J2EE application, you may be able to improve the application's performance by setting Java Performance Options for the JVM running OC4J where your application is deployed.

This section covers the following topics:

- Setting the JVM Heap Size for OC4J Processes
- Setting the JVM Server Option for OC4J Processes
- Setting the JVM AggressiveHeap Option for OC4J Processes
- Setting the JVM Stack Size Option for OC4J Processes
- Setting the JVM Permanent Generation Option for OC4J Processes
- Setting the JVM Thread Synchronization Option for OC4J Processes
- Setting the OC4J DMS Sensors Option
- Setting the OC4J JDBC DMS Statement Metrics Option
- Setting the OC4J Dedicated RMI Context Option
- Setting the OC4J Define Column Type Option
- Using Application Server Control to Change OC4J JVM Command Line Options

When running Oracle Application Server, the module `mod_oc4j` is the connector from Oracle HTTP Server to one or more OC4J Instances. Each OC4J process within an OC4J Instance runs in its own Java Virtual Machine (JVM) and is responsible for parsing J2EE requests and generating a response. When a request comes into Oracle HTTP Server, `mod_oc4j` picks an OC4J process and routes the request to the selected OC4J process. Within each OC4J Instance all of the OC4J JVM processes use the same configuration and start with the same Java options. Likewise, unless a process dies or there is some other problem, each OC4J process that is part of an OC4J Instance has the same J2EE applications deployed to it.

See Also: "Using Application Server Control to Change OC4J JVM Command Line Options" on page 6-11

Setting the JVM Heap Size for OC4J Processes

If you have sufficient memory available on your system and your application is memory intensive, you can improve your application performance by increasing the JVM heap size from the default values. While the amount of heap size required varies based on the application and on the amount of memory available, for most OC4J server applications, a heap size of at least 128 Megabytes is advised. If you have sufficient memory, using a heap size of 256 Megabytes or larger is preferable.

To change the size of the heap allocated to the OC4J processes in an OC4J Instance, use the procedures outlined in "Using Application Server Control to Change OC4J JVM Command Line Options" on page 6-11, and specify the following Java options:

```
-Xmssizem -Xmxsizem
```

Where *size* is the desired Java heap size in megabytes.

If you know that your application will consistently require a larger amount of heap, you can improve performance by setting the minimum heap size equal to the maximum heap size, by setting the JVM `-Xms` size to be the same as the `-Xmx` size.

For example, to specify a heap size of 128 megabytes, specify the following:

```
-Xms128m -Xmx128m
```

You should set your maximum Java heap size so that the total memory consumed by all of the JVMs running on the system does not exceed the memory capacity of your system. If you select a value for the Java heap size that is too large for your hardware configuration, one or more of the OC4J processes within the OC4J Instance may not start, and Oracle Enterprise Manager Application Server Control reports an error. Review the log files for the OC4J Instance in the directory `$ORACLE_HOME/opmn/logs`, to find the error report:

```
Could not reserve enough space for object heap
Error occurred during initialization of VM
```

If you select a value for the JVM heap size that is too small, none of the OC4J processes will be able to start, and Application Server Control reports an error. If you review the log files for the OC4J Instance in the directory `$ORACLE_HOME/opmn/logs`, you may find errors similar to the following:

```
java.lang.OutOfMemoryError
```

Note: There are other reasons why java.lang.OutOfMemoryError error may occur. For example, if the application has a memory leak.

If the system runs out of memory, the OC4J process will shut down. This will happen if references to the objects are not released. For example, if objects are stored in a hash table or vector and never again removed.

It is of course possible that your process actually needs to use a lot of memory. In this case, the maximum heap size for the process should be increased to avoid frequent garbage collection.

To maximize performance, set the maximum heap size to accommodate application requirements. To determine how much Java heap you need, use the JVM metrics `freeMemory` and `totalMemory`. Subtracting the free memory from total memory gives the amount of heap that was consumed. To determine how much Java heap you need in a non-production environment, you can include calls in your program to the `Runtime.getRuntime().totalMemory()` and `Runtime.getRuntime().freeMemory` methods in the `java.lang` package (including these calls in a production environment could have a negative performance impact).

See Also:

- Table A–9, " JVM Metrics (JVM)" on page A-7
- You can find detailed information about JVM options and their impact on performance on the JVM vendor’s Web sites

Setting the JVM Server Option for OC4J Processes

Oracle Application Server 10*g* 9.0.4 uses the `-server` by default on UNIX systems (this is a change from previous Oracle9*i*AS releases). On UNIX systems, Java runs in one of two modes set with the options: `-client` and `-server`. If you need to change this option, use the procedures outlined in "Using Application Server Control to Change OC4J JVM Command Line Options" on page 6-11, and specify the `-client` Java option.

Oracle Application Server 10*g* 9.0.4 uses the 1.4.1 version of the Java virtual machine (JVM), except on Linux, which uses 1.4.2. This JVM version includes an improved JIT compiler from previous JVM releases. Many long-running applications will perform better with the improved JIT. However, due to the increased quality of compilation, applications may experience slower program startup times or occasional pauses in other parts of a program (as compared with older versions of the JVM). In a multi-processor system, the compilation thread runs concurrently with OC4Jstartup, reducing the impact on startup time.

On UNIX systems, using the `–server` option also changes the default heap allocation. For a given heap size, larger allocations are made to the Eden and Survivor generations at the expense of the Old generation. The Permanent generation is not affected. The memory footprint of the heap is not directly affected. See the following site for more details on heap sizes, names, and garbage collection,

`http://wireless.java.sun.com/midp/articles/garbagecollection2/`

Note: The `-server` option or the `–client` option, when used, must be specified first before all other Java options.

Setting the JVM AggressiveHeap Option for OC4J Processes

In the 1.4.2 version of the Java virtual machine (JVM), the `-XX:+AggressiveHeap` option was optimized for long-running, memory allocation-intensive applications. Many applications will exhibit dramatically improved performance and scalability if the `-XX:+AggressiveHeap` option is specified. To set this option, use the procedures outlined in "Using Application Server Control to Change OC4J JVM Command Line Options" on page 6-11.

See the following site for more details on using the `-XX:+AggressiveHeap` option,

`http://java.sun.com/j2se/1.4.2/1.4.2_whitepaper.html#6`

Setting the JVM Stack Size Option for OC4J Processes

Depending on the particular J2EE application, changing the setting of the command line option `-Xss` for the JVM running OC4J may improve performance. To set this option, use the procedures outlined in "Using Application Server Control to Change OC4J JVM Command Line Options" on page 6-11, and specify the `-Xss` Java option.

This option sets the maximum stack size for C code in a thread to *n*. Every thread that is spawned during the execution of the program passed to java has *n* as its C code stack size. The default C code stack size is 512 kilobytes (`-Xss512k`). A value of 64 kilobytes is the smallest amount of C code stack space allowed per thread.

Oracle recommends that you try the following value to improve the performance of your J2EE applications:

```
-Xss128k
```

Setting the JVM Thread Synchronization Option for OC4J Processes

In the 1.4.x version of the JVM for the Solaris Operating System 2.8 and before, many-to-many LWP (lightweight process) synchronization became the default thread model. With JDK 1.4 and Solaris Operating System 2.8, using the one-to-one alternate thread library may give you some performance improvement (set the `LD_LIBRARY_PATH=/usr/lib/LWP` to use this for the Solaris Operating System 2.8). With the Solaris Operating System 2.9, this is the default thread library.

See the following site for more information,

```
http://java.sun.com/docs/hotspot/threads/threads.html
```

It is important to compare results with the various threading options, to select the appropriate one for your applications. For more information, see the following site,

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

Setting the JVM Permanent Generation Option for OC4J Processes

The `MaxPermSize` option defines the size for the permanent generation in the JDK. Since the default value is 64M (Megabytes) in JDK 1.4.x, generally you do not need to change this value, which is used to hold reflective objects of the VM such as class objects and method objects. However, if your applications dynamically generate and load many classes that require a large permanent generation size, you may see `outOfMemory` errors from the JDK even if you have plenty of free memory in the heap (we found this occurs in some JSP implementations). If this occurs, you can change the permanent generation size by setting the `-XX:MaxPermSize` option, as follows:

```
-XX:MaxPermSize=sizem
```

Where *size* is the desired `MaxPermSize` value.

Setting the OC4J DMS Sensors Option

You can disable the collection of most OC4J built-in performance metrics by setting a property for the JVM running OC4J. The default value for the property `oracle.dms.sensors` is `normal`, which enables the collection of built-in performance metrics. You can disable OC4J built-in performance metrics collection by setting the `oracle.dms.sensors` property to the value `none`. For most J2EE applications, using the default value, `normal`, should have minimal impact on performance.

Note: Setting `oracle.dms.sensors` value to `none` causes Oracle Enterprise Manager Application Server Control to report "unavailable" for some values that are based on DMS metrics.

Table 6–2 lists the supported `oracle.dms.sensors` property values.

Table 6–2 DMS Sensor oracle.dms.sensors Property Supported Values

Property Value	Description
`none`	Disable DMS gathering metrics.
`normal`	Enable normal level DMS metrics. This is the default value.
`heavy`	Enable heavy DMS metrics.
`all`	Enable all DMS metrics.

Note: Prior to Oracle Application Server 10*g* (9.0.4), Oracle Application Server used the property `oracle.dms.gate` to enable DMS metrics. Setting this as follows, `oracle.dms.gate=false` is this equivalent to setting `oracle.dms.sensors=none`.

Setting `oracle.dms.gate=true` is equivalent to setting `oracle.dms.sensors=normal`.

Using `oracle.dms.gate` is deprecated in Oracle Application Server 10*g*. This property may not be supported in upcoming releases.

Some Oracle Application Server components that run in OC4J do not use the `oracle.dms.sensors` property to control their DMS metrics. For example, the Portal PPE `web.xml` specified configuration parameter `dmsLogging` controls DMS metric collection for the Portal PPE.

The JDBC drivers also do not use the `oracle.dms.sensors` property to control certain JDBC metrics. To enable the collection of JDBC statement metrics, use the properties, `oracle.jdbc.DMSStatementCachingMetrics` and `oracle.jdbc.DMSStatementMetrics`.

See Also:

- "Setting the OC4J JDBC DMS Statement Metrics Option" on page 6-9
- "Conditional Instrumentation Using DMS Sensor Weight" on page 9-20
- Appendix D, "Configuring the Parallel Page Engine" in *Oracle Application Server Portal Configuration Guide*

Setting the OC4J JDBC DMS Statement Metrics Option

To improve performance, by default OC4J does not collect JDBC statement metrics. The properties, `oracle.jdbc.DMSStatementCachingMetrics` and `oracle.jdbc.DMSStatementMetrics` are by default, set to `false`. When these properties are `false`, performance is improved since OC4J does not collect expensive JDBC statement metrics.

Setting these properties to `true` may have a negative impact on performance. Only set these to `true` when you need to collect JDBC statement metrics.

When `oracle.jdbc.DMSStatementCachingMetrics` property is set to `true` and JDBC statement caching is enabled the JDBC statement metrics are available.

When JDBC statement caching is disabled, make the JDBC statement metrics available by setting the property `oracle.jdbc.DMSStatementMetrics` to `true`.

Disabling these properties by setting the values to `false` only affects the JDBC DMS metrics.

See Also: "Setting the OC4J DMS Sensors Option" on page 6-8

Setting the OC4J Dedicated RMI Context Option

Setting the dedicated RMI context property to `false` using the command line option `-Ddedicated.rmicontext= false` for the OC4J may improve performance when an EJB client is doing multiple initial context lookups within the same JVM.

See Also: "Setting the OC4J Dedicated RMI Context Option for Remote EJB Clients" on page 6-74

Setting the OC4J Define Column Type Option

Set the `DefineColumnType` property to `true` when you are using an Oracle JDBC driver that is prior to Release 9.2 and you are not using statement caching. Setting this option to `true` avoids a round-trip when executing a select over the Oracle JDBC driver.

Note: If you are using a JDBC Driver version 9.2 or above, and you use statement caching, do not set `DefineColumnType` to `true`; this is redundant, since these drivers have similar functionality built-in.

When the driver performs a query, it first uses a round trip to a database to determine the types that it should use for the columns of the result set. Then, when JDBC receives data from the query, it converts the data, as necessary, as it populates the result set. When you specify column types for a query with the

`DefineColumnType` option set to `true`, you avoid the first round trip to the Oracle database.

The default value for `DefineColumnType` is `false`.

Note: This option only applies to EJB CMP entity beans.

If the value of `DefineColumnType` changes, and OC4J is restarted, the updated value only applies to applications deployed after the value is changed.

See Also: "Setting the JDBC Statement Cache Size in Data Sources" on page 6-19

Using Application Server Control to Change OC4J JVM Command Line Options

To change the Java command line options for an OC4J Instance, go to the OC4J instance homepage and perform the following steps:

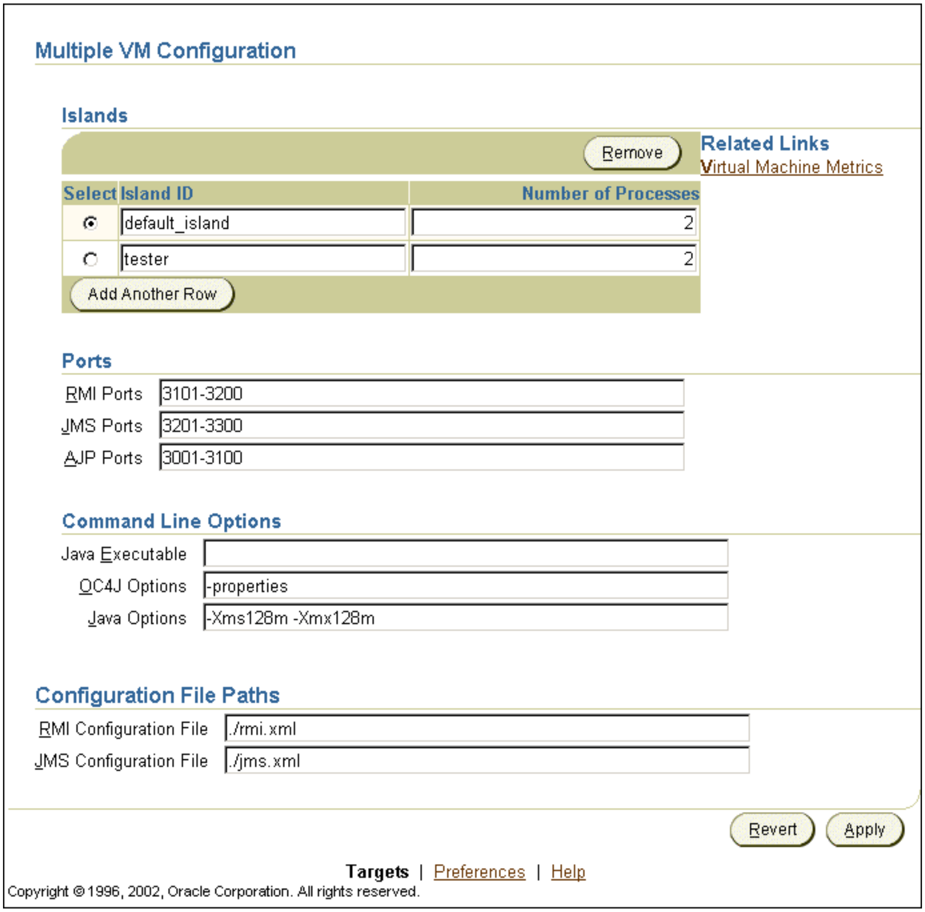
1. Stop the OC4J Instance.
2. Drill down to the Server Properties page.
3. In the Command Line Options area of the Server Properties page, under the heading Multiple VM Configuration, set the Java Options.

   For example, enter the following to set the JVM heap sizes to 128 Megabytes:

   ```
   -Xmx128m
   ```
4. Use the **Apply** button to apply the changes.
5. Start the OC4J Instance.

Figure 6–1 shows the Server Properties page with Java Options.

Figure 6–1 Setting Java Heap Size for an OC4J Instance Using Application Server Control



Setting Up Data Sources – Performance Issues

A data source, which is the instantiation of an object that implements the `javax.sql.DataSource` interface, enables you to retrieve a connection to a database server. This section describes data source configuration options for global data sources. A global data source is available to all the deployed applications in an OC4J Instance.

This section covers the following topics:

- Emulated and Non-Emulated Data Sources
- Using the EJB Aware Location Specified in Emulated Data Sources
- Setting the Maximum Open Connections in Data Sources
- Setting the Minimum Open Connections in Data Sources
- Setting the Cached Connection Inactivity Timeout in Data Sources
- Setting the Wait for Free Connection Timeout in Data Sources
- Setting the Connection Retry Interval in Data Sources
- Setting the Maximum Number of Connection Attempts in Data Sources
- Setting the JDBC Statement Cache Size in Data Sources
- Setting the JDBC Prefetch Size for a CMP Entity Bean
- Using Application Server Control to Change Data Source Configuration Options

---

Note: If your data source is provided by a third party, you may need to set certain properties. These properties should be defined in the third-party documentation.

See Also:

- "Improving EJB Performance in Oracle Application Server" on page 6-40
- *Oracle Application Server Containers for J2EE User's Guide*
- *Oracle Application Server Containers for J2EE Services Guide*
- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

Emulated and Non-Emulated Data Sources

Some of the performance related configuration options have different affects, depending on the type of the data source. OC4J supports two types of data sources, emulated and non-emulated:

The pre-installed default data source is an emulated data source. Emulated data sources are wrappers around Oracle or non-Oracle data sources. If you use these data sources, your connections are extremely fast, because they do not provide full XA or JTA global transactional support. We recommend that you use these data sources for local transactions or when your application requires access or update to a single database. You can use emulated data sources for Oracle or non-Oracle databases.

You can use the emulated data source to obtain connections to different databases by changing the values of the `url` and `connection-driver` parameters.

The following is a definition of an emulated data source:

```
<data-source
  class="com.evermind.sql.DriverManagerDataSource"
  name="OracleDS"
  location="jdbc/OracleCoreDS"
  xa-location="jdbc/xa/OracleXADS"
  ejb-location="jdbc/OracleDS"
  connection-driver="oracle.jdbc.driver.OracleDriver"
  username="scott"
  password="tiger"
  url="jdbc:oracle:thin:@localhost:5521:oracle"
  inactivity-timeout="30"
/>
```

Non-emulated data sources are pure Oracle data sources. These are used by applications that want to coordinate access to multiple sessions within the same database or to multiple databases within a global transaction.

Using the EJB Aware Location Specified in Emulated Data Sources

Each data source is configured with one or more logical names that allow you to identify the data source within J2EE applications. The `ejb-location` is the logical name of an EJB data source. In addition, use the `ejb-location` name to identify data sources for most J2EE applications, where possible, even when not using EJBs. The `ejb-location` only applies to emulated data sources. You can use this option for single phase commit transactions or emulated data sources.

Using the `ejb-location`, the data source manages opening a pool of connections, and manages the pool. Opening a connection to a database is a time-consuming process that can sometimes take longer than the operation of getting the data itself. Connection pooling allows client requests to have faster response times, because the applications do not need to wait for database connections to be created. Instead, the applications can reuse connections that are available in the connection pool.

Note: Oracle recommends that you only use the `ejb-location` JNDI name in emulated data source definitions for retrieving the data source. For non-emulated data sources, you must use the `location` JNDI name.

Setting the Maximum Open Connections in Data Sources

The `max-connections` option specifies the maximum number of open connections for a pooled data source. To improve system performance, the value you specify for the number `max-connections` depends on a combination of factors including the size and configuration of your database server, and the type of SQL operations that your application performs.

The default value for `max-connections` and the handling of the maximum depends on the data source type, emulated or non-emulated.

For emulated data sources, there is no default value for `max-connections`, but the database configuration limits that affect the number of connections apply. When the maximum number of connections, as specified with `max-connections`, are all active, new requests must wait for a connection to be become available. The maximum time to wait is specified with `wait-timeout`.

For non-emulated data sources, there is a property, `cacheScheme`, that determines how `max-connections` is interpreted. Table 6–3 lists the values for the `cacheScheme` property (`DYNAMIC_SCHEME` is the default value for `cacheScheme`).

See Also:

- "Setting the Wait for Free Connection Timeout in Data Sources" on page 6-18
- "Data Sources" in *Oracle Application Server Containers for J2EE Services Guide*

Table 6–3 Non-emulated Data Source cacheScheme Values

Value	Description
FIXED_WAIT_SCHEME	In this scheme, when the maximum limit is reached, a request for a new connection waits until another client releases a connection.
FIXED_RETURN_NULL_SCHEME	In this scheme, the maximum limit cannot be exceeded. Requests for connections when the maximum has already been reached return null.
DYNAMIC_SCHEME	In this scheme, you can create new pooled connections above and beyond the maximum limit, but each one is automatically closed and freed as soon as the logical connection instance is finished being used, where it is returned to the available cache. DYNAMIC_SCHEME is the default value for cacheScheme.

The tradeoffs for changing the value of max-connections are:

- For some applications you can improve performance by limiting the number of connections to the database (this causes the system to queue requests in the mid-tier). For example, for one application that performed a combination of updates and complex parallel queries into the same database table, performance was improved by over 35% by reducing the maximum number of open connections to the database by limiting the value of max-connections.

Note: You should check to make sure that your database is configured to allow at least the total number of open connections, as specified by the data sources max-connections option for all your J2EE applications.

Setting the Minimum Open Connections in Data Sources

The min-connections option specifies the minimum number of open connections for a pooled data source.

For applications that use a database, performance can improve when the data source manages opening a pool of connections, and manages the pool. This can improve performance because incoming requests don't need to wait for a database connection to be established; they can be given a connection from one of the available connections, and this avoids the cost of closing and then reopening connections.

By default, the value of `min-connections` is set to 0. When using connection pooling to maintain connections in the pool, specify a value for `min-connections` other than 0.

For emulated and non-emulated data sources, the `min-connections` option is treated differently.

For emulated data sources, when starting up the initial `min-connections` connections, connections are opened as they are needed and once the `min-connections` number of connections is established, this number is maintained.

For non-emulated data sources, after the first access to the data source, OC4J then starts the `min-connections` number of connections and maintains this number of connections.

Limiting the total number of open database connections to a number your database can handle is an important tuning consideration. You should check to make sure that your database is configured to allow at least as large a number of open connections as the total of the values specified for all the data sources `min-connections` options, as specified in all the applications that access the database.

Note: If the `min-connections` is set to a value other than zero, the specified number of connections will be maintained; OC4J maintains the connections when they are not in use, and they do not time out when the specified `inactivity-timeout` is reached.

Once the specified connections are opened, OC4J does not provide a way to close the connections, except by stopping OC4J.

Setting the Cached Connection Inactivity Timeout in Data Sources

The `inactivity-timeout` specifies the time, in seconds, to cache unused connections before closing them.

To improve performance, you can set the `inactivity-timeout` to a value that allows the data source to avoid dropping and then re-acquiring connections while your J2EE application is running.

The default value for the `inactivity-timeout` is 60 seconds, which is typically too low for applications that are frequently accessed, where there may be some inactivity between requests. For most applications, to improve performance, we recommend that you increase the `inactivity-timeout` to 120 seconds.

To determine if the default `inactivity-timeout` is too low, monitor your system. If you see that the number of database connections grows and then shrinks during an idle period, and grows again soon after that, you have two options: you can increase the `inactivity-timeout`, or you can increase the `min-connections`.

> **See Also:** "Setting the Minimum Open Connections in Data Sources" on page 6-16

Setting the Wait for Free Connection Timeout in Data Sources

The `wait-timeout` specifies the number of seconds to wait for a free connection if the connection pool does not contain any available connections (that is, the number of connections has reached the limit specified with `max-connections` and they are all currently in use).

If you see connection timeout errors in your application, increasing the `wait-timeout` can prevent the errors. The default `wait-timeout` is 60 seconds.

If database resources, including memory and CPU are available and the number of open database connections is approaching `max-connections`, you may have limited `max-connections` too stringently. Try increasing `max-connections` and monitor the impact on performance. If there are not additional machine resources available, increasing `max-connections` is not likely to improve performance.

You have several options in the case of a saturated system:

- Increase the allowable `wait-timeout`.
- Evaluate the application design for potential performance improvements.
- Increase the system resources available and then adjust these parameters.

Setting the Connection Retry Interval in Data Sources

The `connection-retry-interval` specifies the number of seconds to wait before retrying a connection when a connection attempt fails.

If the `connection-retry-interval` is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for the `connection-retry-interval` is 1 second.

Setting the Maximum Number of Connection Attempts in Data Sources

The `max-connect-attempts` option specifies the maximum number of times to retry making a connection. This option is useful to control when the network is not stable, or the environment is unstable for any reason that sometimes makes connection attempts fail.

If the `connection-retry-interval` option is set to a small value, or a large number of connection attempts is specified with `max-connect-attempts` this may degrade performance if there are many retries performed without obtaining a connection.

The default value for `max-connect-attempts` is 3.

Setting the JDBC Statement Cache Size in Data Sources

To lower the overhead of repeated cursor creation and repeated statement parsing and creation, you can use statement caching with database statements. To enable JDBC statement caching, which caches executable statements that are used repeatedly, configure a datasource to use statement caching. A JDBC statement cache is associated with a particular physical connection maintained by a datasource. A statement cache is not per datasource so it is not shared across all physical connections. The JDBC statement cache is maintained in the middle-tier (not in the database server).

You can dynamically enable and disable statement caching programmatically using the `setStmtCacheSize()` method on the connection object.

To configure JDBC statement caching for a datasource, use the `stmt-cache-size` attribute to set the size of the cache. This attribute sets the maximum number of statements to be placed in the cache. If you do not specify the `stmt-cache-size` attribute or set it to zero, the statement cache is disabled.

The following XML sets the statement cache size to 200 statements.

```
<data-source>
 ...
stmt-cache-size="200"
</data-source>
```

To set the `stmt-cache-size` attribute, first determine how many distinct statements the application issues to the database. Then, set the size of the cache to this number. If you do not know the number of statements that your application issues to the database, you can use the JDBC performance metrics to assist you with determining the statement cache size. To use the statement metrics you need to set the Java property `oracle.jdbc.DMSStatementMetrics` to `true` for the OC4J.

See Also:

- "JDBC Data Source Statement Metrics" on page A-11
- *Oracle9i JDBC Developer's Guide and Reference*

Setting the JDBC Prefetch Size for a CMP Entity Bean

You can use the `prefetch-size` parameter to change the data source behavior for a JDBC query from a CMP Entity bean. However, this parameter is configured in `orion-ejb-jar.xml` rather than in `data-sources.xml`.

See Also: "Configuring Parameters for CMP Entity Beans" on page 6-43

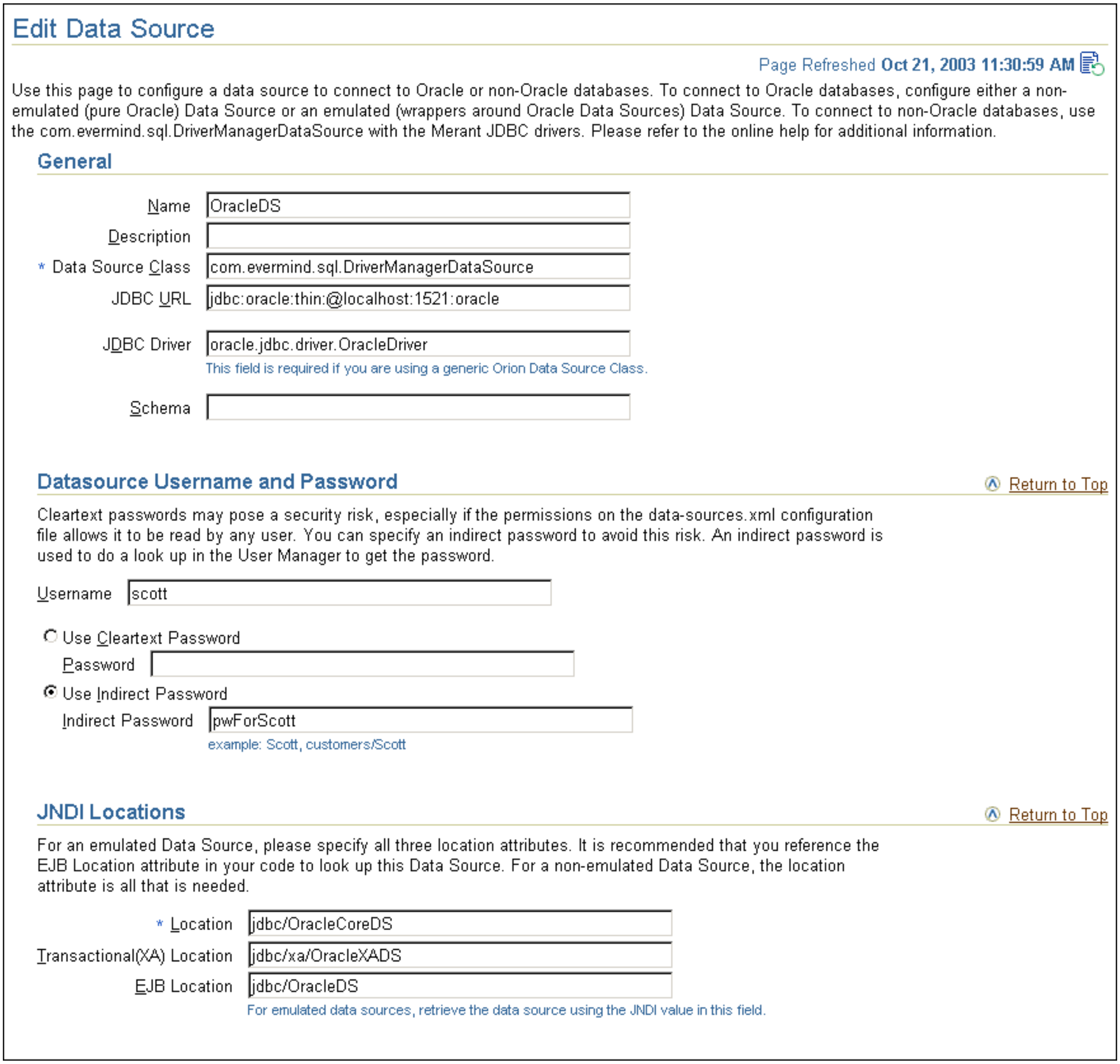
Using Application Server Control to Change Data Source Configuration Options

Figure 6–2 shows the Oracle Enterprise Manager Application Server Control configuration page that lets you view or modify a data source. This page is available in Application Server Control by selecting the Edit button for a selected data source from the Data Sources page from the application default page for an OC4J Instance, or by selecting data sources from the administration section of a deployed application's description page (this is only available when the application has its own local data source).

Application Server Control stores the data sources elements that you add or modify in an XML file. This file defaults to the name `data-sources.xml` and is located in `/j2ee/home/config`. If you want to change the name or the location of this file, you can do this in the General Properties page off of the default application screen or off of your specific application's page, when the application specifies a local data source.

Note: You can also use the Application Server Control Advanced Properties links to create or edit data sources. This allows you to add data sources using the XML definitions which is useful if you have been provided the XML.

Figure 6–2 Application Server Control Data Sources Configuration Page



Setting server.xml Configuration Parameters

This section covers parameters that you can tune for OC4J performance in the `server.xml` file for an OC4J Instance.

This section covers the following topics:

- Setting the OC4J Transaction Configuration Timeout in server.xml
- Setting the OC4J Task Manager Granularity in server.xml
- Setting the OC4J Task Manager Check for Updates Control in server.xml
- Setting the OC4J Global Thread Pool Attributes in server.xml
- Setting the OC4J Options for Stateful Session Bean Passivation in server.xml
- Using Application Server Control to Change server.xml Configuration Options

Setting the OC4J Transaction Configuration Timeout in server.xml

You can change the default value for the transaction configuration timeout in the `transaction-config` element in the `server.xml` file for the OC4J Instance. This configuration parameter specifies the maximum time taken for a transaction to finish before it can get rolled back due to a timeout, and applies to all transactions on the OC4J Instance.

By default `server.xml` sets the `transaction-config` to 30000 (30 seconds). You can change the `transaction-config timeout` value for applications that are getting transaction timeout errors, or if you know the transactions take longer than 30 seconds (including the time for waiting for connections set by `wait-timeout` in `datasources.xml`).

To change the `transaction-config timeout` value, change the following line in `server.xml`. For example, the following line in `server.xml` sets the `transaction-config timeout` parameter to 60 seconds:

```
<transaction-config timeout="60000"/>
```

Note: The `transaction-config` timeout is not an EJB specific timeout, but affects all transactions which use EJBs. However, the timeout specified with the `transaction-config timeout` value set in `server.xml` does not apply to MDB transactions.

The `transaction-config` timeout applies for all transactions running in OC4J, and therefore must be big enough for your longest transaction. If you specify a small timeout value for `transaction-config` timeout, then you cannot set the timeout to a larger value for an individual EJB, since the `transaction-config timeout` applies for all transactions at the EJB level. Thus, the timeout should be set to a value greater than or equal to the timeouts used within a transaction (for example the data sources `wait-timeout`, and the EJB `call-timeout`).

See Also:

- "Setting the Wait for Free Connection Timeout in Data Sources" on page 6-18
- "Configuring Parameters that Apply for All EJBs (Except MDBs)" on page 6-41
- "Configuring Parameters for Message Driven Beans (MDBs)" on page 6-56

Setting the OC4J Task Manager Granularity in server.xml

The OC4J task manager is an OC4J background process that performs cleanup operations, including the task of timing out HttpSessions. The task manager runs at regular intervals. Using the `taskmanager-granularity` attribute in `server.xml`, you can manage when the task manager runs. This attribute sets how often the task manager performs its cleanup operations. The value specified is in milliseconds and the default value is 1000 milliseconds.

The default `taskmanager-granularity` interval is usually adequate. If you modify the default value and set the value too high, such as to a value greater than 60000, one minute, this can delay the task of timing out of HttpSessions, which could lead to an `OutOfMemory` error (if you use HttpSessions).

For example, the following entry in `server.xml` sets `taskmanager-granularity` to 60000 milliseconds (1 minute).

```
<application-server ...  taskmanager-granularity="60000" ...>
```

Note: Changing the `taskmanager-granularity` can affect the timing and accuracy for some of the EJB Entity and Session Bean parameters. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for complete details.

Setting the OC4J Task Manager Check for Updates Control in server.xml

The OC4J task manager is an OC4J background process that performs cleanup. The task manager `application-server` element uses the boolean attribute `check-for-updates` to control if the task manager checks the .xml configuration files for changes when the task manager executes.

> **Note:** The attribute `check-for-updates` is only used for standalone OC4J.

The default value for `check-for-updates` is `true`, which means when you update an xml file OC4J incorporates the changes when the task manager runs.

When the `application-server` attribute `check-for-updates` is set to `false`, OC4J does not automatically check for updates to xml configuration files. When you want to update xml files with this attribute set to `false`, you need to run the command `admin.jar -updateConfig`. The `updateConfig` flag informs OC4J to check the xml config files for updates.

Setting the OC4J Global Thread Pool Attributes in server.xml

The OC4J `server.xml` global thread pool attributes let you limit the number of threads that OC4J uses and provides a way to customize the OC4J thread pool.

> **Note:** If you do not specify `<global-thread-pool>` in `server.xml`, then the number of threads that OC4J can create is unbounded. With the default behavior, unbounded threads, threads are created on demand as needed.

Table 6–4 shows the `<global-thread-pool>` attributes. You can use these attributes to control the global thread pool as follows:

- You can configure `<global-thread-pool>` in `server.xml` so that OC4J only uses a single thread pool. To use a single thread pool configure only the `min`, `max`, `queue`, and `keepAlive` attributes for `<global-thread-pool>`. Using a single thread pool lets you limit the overall OC4J concurrency within the JVM.
- You can configure `<global-thread-pool>` so that OC4J uses two thread pools. The second thread pool limits the number of threads dedicated to concurrent ORMI connections or other internal connection processing. Note

that the actual processing of requests that originate from these connections is handled by the first thread pool.You only need to configure two thread pools if you have many concurrent remote EJB clients.

To use two thread pools, configure the `min`, `max`, `queue`, and `keepAlive` attributes and the `cx-min`, `cx-max`, `cx-queue`, and `cx-keepAlive` attributes for `<global-thread-pool>` in `server.xml`.

To configure two thread pools, you must also configure all of the following attributes: `min`, `max`, `queue`, and `keepAlive`. If any of these attributes is not configured, OC4J gives the following error message at startup:

```
Error initializing server: Invalid Thread Pool parameter: null
```

Table 6–4 server.xml <global-thread-pool> Attributes

Attribute	Description
`min`	Specifies the minimum number of threads that OC4J can simultaneously execute. By default, a minimum number of threads are preallocated and placed in the thread pool when the container starts. Default Value: 20 (the minimum value is 10)
`max`	Specifies the maximum number of threads that OC4J can simultaneously execute. New threads are spawned, when the queue is full, if the maximum size is not reached and if there are no idle threads. Idle threads are used first before a new thread is spawned. Default Value: 40
`queue`	Specifies the maximum number of requests that can be kept in the queue. Default Value: 80
`keepAlive`	Specifies the number of milliseconds to keep a thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains active. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To never destroy threads, set this timeout to a negative one (-1). Default Value: 600000 (milliseconds)
`cx-min`	Specifies the minimum number of ORMI connection threads from EJB clients plus other internal connection or listener threads that OC4J can simultaneously execute. Default Value: 20 (the minimum value is 10)
`cx-max`	Specifies the maximum number of ORMI connection threads from EJB clients plus other internal connection or listener threads that OC4J can simultaneously execute. Default Value: 40

Table 6–4 (Cont.) server.xml <global-thread-pool> Attributes

Attribute	Description
`cx-queue`	Specifies the maximum number of ORMI connection requests that can be kept in the queue. Value is an integer. Default Value: 80
`cx-keepAlive`	Specifies the number of milliseconds to keep a connection thread alive (idle) while waiting for a new request. This timeout designates how long an idle thread remains active. If the timeout is reached, the thread is destroyed. The minimum time is a minute. Time is set in milliseconds. To specify that threads are never destroyed, set the `cx-timeout` to negative one (-1). Default Value: 600000 milliseconds.
`debug`	When set to true, print the application server thread pool information at startup. Valid values: `true`, `false` Default Value: `false`

Recommendations for specifying attributes with `<global-thread-pool>`.

1. To limit resources using OC4J with Oracle Application Server, we recommend using the Oracle HTTP Server `MaxClients` directive instead of setting `<global-thread-pool>` attributes. When OC4J runs in an Oracle Application Server environment, `mod_oc4j` works with OC4J to control OC4J concurrency. In this environment, limiting the number of threads can cause resource contention issues that may result in deadlocks.
2. If you use `<global-thread-pool>` in an Oracle Application Server environment with `mod_oc4j`, we recommend that you set the `mod_oc4j Oc4jCacheSize` directive to 0 to disable persistent connections between `mod_oc4j` and OC4J instances. Note that setting the `Oc4jCacheSize` to 0 may degrade performance.
3. When using `<global-thread-pool>` to limit concurrency in a standalone OC4J environment, set the `max` value to handle the desired number of concurrent requests. The value specified for the `max` attribute specifies the overall concurrency level; set this value to include 10 internal threads, plus the desired concurrency for the applications that run on OC4J (number of threads), plus, when using MDBs, one thread per MDB listener. You may want to use the `<global-thread-pool>` with standalone OC4J when you have limited resources and many concurrent requests. In this case, specifying `<global-thread-pool>` attributes may improve application performance. If you have

many remote ORMI clients, use 2 thread pools (OC4J uses the second thread pool to handle the remote connections plus 10 internal threads).

4. The `cx-min` and `cx-max` sets the minimum and maximum thread pool size for the ORMI connection threads plus 10 internal listener threads. Thus, set the parameter `cx-max` to the maximum number of ORMI physical connections you expect to have, and add an additional 10 for internal threads. Note that the number of expected ORMI physical connections may be different than the number of active clients.

5. In a production environment, once you figure out the desired number of threads, set the `min` to the same value as the `max`, and set the `keepAlive` attribute to -1. If you choose to set the `min` to a value that is less than `max`, we recommend that you set the `queue` size small (equal to at most two times the size of `max`). We include these recommendations because the pool threads will only grow when the queue, as specified with the `queue` parameter is full, and not before.

   Generally you may also want to set the `cx-min` and `cx-max` to handle transient bursts of connections (when you want the threads to go away between bursts). For this type of configuration, set the `cx-min` equal to your typical connection load, and `cx-max` equal to the maximum expected connections, and set the `cx-queue` size to a small value, for example 2. We include these recommendations because the pool threads will only grow when the queue, as specified with the `cx-queue` parameter is full, and not before.

The following `server.xml` example initializes two thread pools for the OC4J process and prints thread pool information at startup:

```
<application-server ...>
<global-thread-pool min="100" max="100" queue="200"
keepAlive="700000" cx-min="100" cx-max="100" cx-queue="200"
cx-keepAlive="700000" debug="true"/>
.
.
.
</application-server>
```

Setting the OC4J Options for Stateful Session Bean Passivation in server.xml

OC4J automatically performs passivation of stateful session beans unless you set the `enable-passivation` attribute for the element `<sfsb-config>` to `false`.

The default value for the attribute `enable-passivation` is `true`, which means that stateful session bean passivation occurs. If you have a situation where stateful session beans are not in a state to be passivated, set this attribute to `false`.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

Using Application Server Control to Change server.xml Configuration Options

To update and configure values for options in the `server.xml` file, using Application Server Control, first select the OC4J instance you want to modify. Then, select the Administration link and select the Advanced Properties link from the Instance Properties area. On the Advanced Server Properties page, select the `server.xml` link. On the edit `server.xml` page, select and modify the elements and attributes that you need to change. Finally, select the **Apply** button to apply the changes.

If you do not use Application Server Control, then edit `server.xml` in the `$ORACLE_HOME/j2ee/`*instance_name*`/config` directory, and use the `dcmctl` command to update the Oracle Application Server configuration as follows:

```
% dcmctl updateconfig -ct oc4j
% dcmctl restart -ct oc4j
```

Improving Servlet Performance in Oracle Application Server

This section discusses configuration options and performance tips specific to servlets for optimizing OC4J performance.

This section covers the following topics:

- Improving Performance by Altering Servlet Configuration Parameters
- Servlet Performance Tips

Improving Performance by Altering Servlet Configuration Parameters

This section covers the following:

- Loading Servlet Classes at Startup
- Reducing Requests for Static Pages and Images
- Setting the Servlet Session Timeout

Loading Servlet Classes at Startup

By default, OC4J loads a servlet when the first request is made. OC4J also lets you load servlet classes when the JVM that runs the servlet is started. To do this, add the `<load-on-startup>` sub-element to the `<servlet>` element in the application's `web.xml` configuration file.

Using the load-on-startup facility increases the start-up time for your OC4J process, but decreases first-request **latency** for servlets.

For example, add the `<load-on-startup>` as follows:

```
<servlet>
      <servlet-name>viewsrc</servlet-name>
      <servlet-class>ViewSrc</servlet-class>
      <load-on-startup> </load-on-startup>
</servlet>
```

Using Application Server Control you can specify that OC4J load an entire Web Module on startup. To specify load on startup, select the Website Properties page for an OC4J Instance and then use the Load on Startup checkbox.

Reducing Requests for Static Pages and Images

This `<expiration-setting>` element, that can be set in either `global-web-application.xml` or `orion-web.xml` sets the expiration for a given set of resources. This element can reduce the requests to the server by asking the browser to cache certain requests. If the Oracle Application Server instances uses OracleAS Web Cache, then this element is less useful, since Web Cache should serve such requests, when it is used. The `<expiration-setting>` determines how long before resources expire in the browser. The browser reloads an expired resource upon the next request for it.

This option is useful for setting caching policies, such as for not reloading images as frequently as documents.

To set the `<expiration-setting>` element, use the following attributes: `expires`, `url-pattern`.

- `expires` specifies the number of seconds before expiration, or when set to "`never`" specifies no expiration. The default setting for `expires` is "0" (zero), for immediate expiration.
- `url-pattern` specifies a URL pattern that the expiration applies to. For example, `url-pattern="*.gif"`

Setting the Servlet Session Timeout

The default servlet session timeout for OC4J is 20 minutes. You can change this for a specific application by setting the `<session-timeout>` parameter in the `<session-config>` element of `web.xml`. If this value is set too low, you may loose your saved session before getting the chance to reuse it. If this value is set too high, you may save too much session state and consume too much memory. The amount of memory used in each session depends on the size of the objects the application creates and puts into the sessions. Setting either a too small value, or a too large value for the session timeout can have an impact on performance.

Servlet Performance Tips

The following tips can enable you to avoid or debug potential performance problems:

- Analyze Servlet Duration
- Understand Server Request Load
- Find Large Servlets That Require a Long Load Time
- Watch for Unused Sessions and Session Invalidation
- Load Servlet Session Security Routines at Startup

Analyze Servlet Duration

It is useful to know the average duration of the servlet (and JSP) requests in your J2EE enterprise application. By understanding how long a servlet takes when the system is not under load, you can more easily determine the cause of a performance problem when the system is loaded. The average duration of a given servlet is reported in the metric `service.avg` for that servlet. You should only examine this value after making many calls to the servlet so that any startup overhead such as class loading and database connection establishment will be amortized.

As an example, suppose you have a servlet for which you notice the `service.avg` is 32 milliseconds. And suppose you notice a response time increase when your system is loaded, but not CPU bound. When you examine the value of `service.avg`, you might find that the value is close to 32 ms, in which case you can assume the degradation is probably due to your system or application server configuration rather than in your application. If on the other hand, you notice that `service.avg` has increased significantly, you may look for the problem in your application. For example, multiple users of the application may be contending for the same resources, including but not limited to database connections.

See Also: Table A–16, " OC4J/application/WEBs Metrics"

Understand Server Request Load

In debugging servlet and JSP problems, it is often useful to know how many requests your OC4J processes are servicing. If the problems are performance related, it is always helpful to know if they are aggravated by a high request load. You can track the requests for a particular OC4J Instance using Application Server Control, or by viewing the application's web module metrics.

See Also: Table A–16, " OC4J/application/WEBs Metrics"

Find Large Servlets That Require a Long Load Time

You may find that a servlet application is especially slow the first time it is used after the server is started, or that it is intermittently slow. It is possible that when this happens the server is heavily loaded, and the response time is suffering as a result. If there is no indication of a high load, however, which you can detect by monitoring your access logs, periodically monitoring CPU utilization, or by tracking the number of users that have active requests on the HTTP server and OC4J, then you may just have a large servlet that takes a long time to load.

You can see if you have a slow loading servlet by looking at `service.maxTime`, `service.minTime`, and `service.avg`. If the time to load the servlet is much higher than the time to service, the first user that accesses the servlet after your system is started will feel the impact, and `service.maxTime` will be large. You can avoid this by configuring the system to initialize your servlet when it is started.

See Also: "Loading Servlet Classes at Startup" on page 6-29

Watch for Unused Sessions and Session Invalidation

You should regularly monitor your applications looking for unused sessions. It is easy to inadvertently write servlets that do not invalidate their sessions. Without source code for the application software, you may not know this could be a problem on your host, but sooner or later you would notice a higher consumption of memory than expected. You can see if there are sessions which are not utilized or sessions which are not being properly invalidated after being used with the session metrics, including: `sessionActivation`, `sessionActivation.completed` and `sessionActivation.active`.

JSPs by default create sessions. If you do not need to use sessions in your JSPs, turn them off.

The following example shows an application that creates sessions, but never uses them. In this example, we show metrics from a JSP under `/oc4j/`*`application`*`/WEBs/`*`context`*:

```
session.Activation.active:       500 ops
session.Activation.completed:      0 ops
```

This application created 500 sessions and all are still active. Possibly, this indicates that the application makes unnecessary use of the sessions and it is just a matter of time before this will cause memory or CPU consumption problems.

A well-tuned application shows `sessionActivation.active` with a value that is less than `sessionActivation.completed` before the session timeout. This indicates that the sessions are probably being used and cleaned up.

Suppose we have a servlet that uses sessions effectively and invalidates them appropriately. Then we might see a set of metrics such as the following, under `/oc4j/`*`application`*`/WEBs/`*`context`*:

```
session.Activation.active:         2 ops
session.Activation.completed:    500 ops
```

The fact that two sessions are active and more than 500 have been created and completed indicates that sessions are being invalidated after use.

See Also:

- "Impact of Session Management on Performance" on page 6-36
- Table A–17, " OC4J/`application`/WEBs/`context` Metrics"

Load Servlet Session Security Routines at Startup

OC4J uses the class `java.security.SecureRandom` for secure seed generation. The very first call to this method is time consuming. Depending on how your system is configured for security, this method may not be called until the very first request for a session-based servlet is received by the Application Server. One alternative is to configure the application to load-on-startup in the application's `web.xml` configuration file and to create an instance of `SecureRandom` during the class initialization of the application. The result will be a longer startup time in lieu of a delay in servicing the first request.

See Also: "Loading Servlet Classes at Startup" on page 6-29

Improving JSP Performance in Oracle Application Server

OracleJSP is Oracle's implementation of the Sun Microsystems JavaServer Pages specification. Some of the additional features it includes are custom JavaBeans for accessing Oracle databases, SQL support, and extended datatypes.

This section explains how you can improve OracleJSP performance. It contains the following topics:

- Improving Performance by Altering JSP Configuration Parameters
- Improving Performance by Tuning JSP Code

Note: A JSP is translated into a Java servlet before it runs, therefore servlet performance issues also apply for JSPs.

Oracle Application Server provides JSP tag libraries that include some features that may improve the performance of J2EE applications. For example, you may be able to use the JSP caching features available in the tag libraries to increase the speed and scalability for your applications:

- The JESI tag library supports the use of Oracle Application Server Web Cache. This supports the use of the HTTP-level cache, maintained outside the application, that provides very fast cache operations. Oracle Application Server Web Cache is capable of caching static data, such HTML, GIF, or JPEG files, or dynamic data, such as servlet or JSP results.
- The Web Object Cache tag library let you capture intermediate results of JSP and servlet execution, and subsequently reuse these cached results in other parts of the Java application logic.

See Also:

- *Oracle Application Server Containers for J2EE Servlet Developer's Guide*
- *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*

Improving Performance by Altering JSP Configuration Parameters

This section describes JSP configuration parameters that you can alter to improve and control JSP operation. These parameters are set for each OC4J Instance by altering the file `global-web-application.xml`.

This section covers the following topics:

- Using the main_mode Parameter
- Additional JSP and OC4J Configuration Parameters

See Also:

- *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* for information on JSP configuration parameters
- *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for information on `global-web-application.xml`

Using the main_mode Parameter

The `main_mode` parameter determines whether classes are automatically reloaded or JSPs are automatically recompiled, in case of changes.

Table 6–1 shows the supported settings for `main_mode`.

Table 6–5 JSP main_mode Parameter Values

Option	Description
`justrun`	The runtime dispatcher does not perform any timestamp checking, so there is no recompilation of JSPs or reloading of Java classes. This mode is the most efficient mode for a deployment environment, where code will not change. If comparing timestamps is unnecessary, as is the case in a typical production deployment environment where source code will not change, you can avoid all timestamp comparisons and any possible retranslations and reloads by setting the `main_mode` parameter to the value `justrun`. Using this value can improve the performance of JSP applications. Note: before you set `main_mode` to the value `justrun`, make sure that the JSP is compiled at least once. You can compile the JSP by invoking it through a browser, or by running your application (using the default value for `main_mode`, `recompile`). This assures that the JSP is compiled before you set the `justrun` flag.
`reload`	The dispatcher will check if any classes have been modified since loading, including translated JSPs. JavaBeans invoked from pages, and any other dependency classes.
`recompile`	This is the default value for `main_mode`. The dispatcher will check the timestamp of the JSP, retranslate it if it has been modified since loading, and execute all `reload` functionality as well.

Note the following when working with the `main_mode` parameter:

- Because of the usage of in-memory values for class file last-modified times, removing a page implementation class file from the file system will *not* by itself cause retranslation of the associated JSP source.
- The page implementation class file will be regenerated when the memory cache is lost. This happens whenever a request is directed to this page after the server is restarted or after another page in this application has been retranslated.
- A page is *not* reloaded just because a statically included file has changed. Statically included files, included through `<%@ include ... %>` syntax as opposed to `<jsp:include ... />` syntax, are included during translation-time.

Note: Before you set `main_mode` to the value `justrun`, make sure that the JSP is compiled at least once. You can compile the JSP by invoking it through a browser, or by running your application.

Additional JSP and OC4J Configuration Parameters

The *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* includes information on additional configuration parameters that affect JSP performance, including the following:

- `check_page_scope`
- `precompile_check`
- `reduce_tag_code`
- `static_text_in_chars`
- `simple-jsp-mapping`
- `tags_reuse_default`
- `enable-jsp-dispatcher-shortcut`

See Also: The section, "JSP Performance Considerations" in *Oracle Application Server Containers for J2EE User's Guide*

Improving Performance by Tuning JSP Code

This section describes changes you can make to your JSP code to improve performance.

This section covers the following topics:

- Impact of Session Management on Performance
- Using Static Template Text Instead of out.print for Outputting Text
- Performance Issues for Buffering JSPs
- Using Static Versus Dynamic Includes

Impact of Session Management on Performance

In general, sessions add performance overhead for your Web applications. Each session is an instance of the `javax.servlet.http.HttpSession` class. The amount of memory per session depends on the size of the objects the application creates and puts into the sessions. You can turn off sessions for your JSPs if you do not want a new session created for each request. By default, in OracleJSP sessions are enabled. If you do not need to use sessions in your JSPs, turn them off by including the following line at the top of the JSP:

```
<%@ page session="false" %>
```

If you use sessions, ensure that you explicitly cancel the session. If you do not cancel a session, it remains active until it times out. Invoke the `invalidate()` method to cancel a session.

The default session timeout for OC4J is 20 minutes. You can change this for a specific application by setting the `<session-timeout>` parameter in the `<session-config>` element of `web.xml`.

For example, the following code shows how you would cancel a session after you have finished using it:

```
HttpSession session;
session = httpRequest.getSession(true);
.
.
.
session.invalidate();
```

OC4J uses the class `java.security.SecureRandom` for secure seed generation. The very first call to this method is time consuming. Depending on how your system is configured for security, this method may not be called until the very first request for a session-based JSP is received by the Application Server. One alternative is to force this call to be made on startup by including a call in the class initialization for some application that is loaded on startup. The result will be a longer startup time in lieu of a delay in servicing the first request.

Note: JSP pages by default use sessions while servlets by default do not use sessions.

See Also:

- *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide* for information on sessions
- *Oracle Application Server Containers for J2EE Servlet Developer's Guide* for information on sessions

Using Static Template Text Instead of out.print for Outputting Text

Using the JSP code `out.print("<html>")` requires more resources than including static template text. For performance reasons, it is best to reserve the use of the `out.print()` command for dynamic text.

Example 6–1 and Example 6–2 are two HTML coding examples. For these JSP samples, Example 6–2 should be more efficient and give better performance.

Example 6–1 Using out.print

```
<%
  out.print("<HTML> <HEAD> <TITLE>Bookstore Home Page</TITLE></HEAD>\n");
  out.print("<BODY BGCOLOR=\"#ffffff\">\n");
  out.print("<H1 ALIGN=\"center\">Book Store Web Commerce Test</H1>\n");
  out.print("<P ALIGN=\"CENTER\">\n");
  out.print("<IMG SRC=\"../bookstore/Images/booklogo.gif\" ALIGN=\"BOTTOM\""+
            "BORDER=\"0\" WIDTH=\"288\" HEIGHT=\"67\"></P>\n");
  out.print("<H2 ALIGN=\"center\">Home Page</H2>\n");
%>
<jsp:useBean  id="randomid"  class="bookstore.BOOKS_Util" scope="request" >
<%
  random_id = randomid.getRandomI_ID();
%>
```

Example 6–2 Using Static Text

```
<HTML> <HEAD> <TITLE>Bookstore Home Page</TITLE></HEAD>
<BODY BGCOLOR=\"#ffffff\">
<H1 ALIGN=\"center\">Bookstore Web Commerce Test </H1>
<P ALIGN=\"CENTER\">
<IMG SRC=\"../bookstore/Images/booklogo.gif\" ALIGN=\"BOTTOM\""+
         "BORDER=\"0\" WIDTH=\"288\" HEIGHT=\"67\"></P>
<H2 ALIGN=\"center\">Home Page</H2>
<jsp:useBean  id="randomid"  class="bookstore.BOOKS_Util" scope="request" >
<%
  random_id = randomid.getRandomI_ID();
%>
```

Performance Issues for Buffering JSPs

By default, a JSP uses an area of memory known as a page buffer. The page buffer, set to 8KB by default, is required if the JSP uses dynamic globalization, `contextType` settings, error pages, or forwards. If the page does not use these features, then you can disable buffering with the following command:

```
<%@ page buffer="none" %>
```

Disabling buffering by setting the buffer value to `none` improves the performance of the page by reducing memory usage and saving the processing step of copying the buffer.

When you need buffering, it is important to select an adequate size for your buffer. If you are writing a page that is larger than the default 8KB buffer, and you have not reset the buffer size, then the JSP `autoflush` will be activated which could have performance implications. Therefore, if buffering is necessary for your JSP, make sure to set the page buffer to an appropriate size. For example, to set the buffer size to 24KB, use the following command:

```
<%@ page buffer="24KB" %>
```

Using Static Versus Dynamic Includes

The `include` directive makes a copy of the included page and copies it into a JSP (`including page`) during translation. This is known as a **static include** (or **translate-time include**) and uses the following syntax:

```
<%@ include file="/jsp/userinfopage.jsp" %>
```

Alternatively, the `jsp:include` tag dynamically includes output from the included page within the output of the including page, during runtime. This is known as a **dynamic include** (or **runtime include**) and uses the following syntax:

```
<jsp:include page="/jsp/userinfopage.jsp" flush="true" />
```

If you have static text, that is not too large, for performance reasons, it is better to use a static include rather than a dynamic include.

In general, when working with includes, note the following:

- Static includes affect page size. Static includes avoid the overhead of the request dispatcher that a dynamic include necessitates, but may be problematic where large files are involved. Static includes are typically used to include small files whose content is used repeatedly in multiple JSPs. For example:
  - Statically include a logo or copyright message at the top or bottom of each page in your application.
  - Statically include a page with declarations or directives, such as imports of Java classes, that are required in multiple pages.
  - Statically include a central `status checker` page from each page of your application.
- Dynamic includes affect processing overhead and performance. Dynamic includes are useful for modular programming. You may have a page that sometimes executes on its own but sometimes is used to generate some of the output of other pages. Dynamically included pages can be reused in multiple including pages without increasing the size of the including pages.

Note: Both static includes and dynamic includes can be used only between pages in the same servlet context.

See Also: *Oracle Application Server Containers for J2EE Support for JavaServer Pages Developer's Guide*

Performance Issues for Including Static Content

JSPs containing a large amount of static content, including large amounts of HTML code that does not change at runtime, may result in slow translation and execution.

There are two workarounds for this issue that may improve performance:

- Put the static HTML into a separate file and use a dynamic `include` command (`jsp:include`) to include its output in the JSP output at runtime.

> **Note:** A static `<%@ include... %>` command would not work. It would result in the included file being included at translation time, with its code being effectively copied back into the including page. This would not solve the problem.

- Put the static HTML into a Java resource file.

  The JSP translator will do this for you if you enable the `external_resource` configuration parameter.

  For pre-translation, the `-extres` option of the `ojspc` tool also offers this functionality.

> **Note:** Putting static HTML into a resource file may result in a larger memory footprint than the preceding `jsp:include` workaround mentioned, because the page implementation class must load the resource file whenever the class is loaded.

Improving EJB Performance in Oracle Application Server

This section covers configuration parameters that you set to control how OC4J handles EJBs. Tuning these options can improve the performance of EJBs running on OC4J.

This section includes the following topics:

- Configuring Parameters that Apply for All EJBs (Except MDBs)
- Configuring Parameters for CMP Entity Beans
- Configuring Parameters for BMP Entity Beans
- Configuring Parameters for Session Beans
- Configuring Parameters for Message Driven Beans (MDBs)

Configuring Parameters that Apply for All EJBs (Except MDBs)

Table 6–6 lists parameters that you can tune for EJB performance that are specific to OC4J. These parameters apply for all types of EJBs, including session and entity beans (except MDBs).

Table 6–6 shows parameters that are specified in `orion-ejb-jar.xml`.

This section also covers the following topic:

- EJB Timeouts Using a Non-default taskmanager-granularity

Table 6–6 EJB Parameters That Apply for All EJB Types (Except MDBs)

Parameter	Description
`call-timeout`	Applies for session and entity beans. This parameter specifies the maximum time to wait for any resource that the EJB container needs, excluding database connections, before the container calls the EJB method. The container throws a `TimedOutException` when the wait time for a resource exceeds the specified `call-timeout` time. Setting the `call-timeout` to a value <=0 specifies an unlimited `call-timeout` (unlimited wait time for resources). Note: if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this causes the `call-timeout` to be calculated based on the new taskmanager-granularity. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for details. Default Value: 90000 milliseconds See Also: "Setting the OC4J Transaction Configuration Timeout in server.xml" on page 6-22
`max-instances`	The number of bean instances allowed in memory – either instantiated or pooled. When this value is reached, the container attempts to passivate the oldest bean instance from memory (this passivation only applies for stateful session beans). If unsuccessful, the container waits the number of milliseconds set in the `call-timeout` attribute to see if a bean instance is removed from memory, either through passivation, using the `remove()` method, or by bean expiration before a `TimeoutExpiredException` is thrown back to the client. To allow an unlimited number of bean instances, set `max-instances` to 0. Default Value: 0 (unlimited)

Table 6–6 (Cont.) EJB Parameters That Apply for All EJB Types (Except MDBs)

Parameter	Description
`max-tx-retries`	Applies for session and entity beans. This parameter specifies the number of times to retry a transaction that was rolled back due to system-level failures. Generally, we recommend that you start by setting `max-tx-retries` to 0 and adding retries only where errors are seen that could be resolved through retries. For example, if you are using serializable isolation and you want to retry the transaction automatically if there is a conflict, you might want to use retries. However, if the bean wants to be notified when there is a conflict, then in this case, you should set `max-tx-retries=0`. Default Value: 0 (for session beans and entity beans) See Also: "Setting the OC4J Transaction Configuration Timeout in server.xml" on page 6-22 See Also: "Setting the Connection Retry Interval in Data Sources" on page 6-18
`min-instances`	The minimum number of bean implementation instances to be kept instantiated or pooled. These instances are created when an EJB of the specified type is accessed, when the first instance is requested, and not at OC4J startup. Default Value: 0 (instances)

EJB Timeouts Using a Non-default taskmanager-granularity

There are EJB administrative tasks that are run at an interval, the length of which depends on the taskmanager granularity. Therefore, if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this change also impacts the interval at which EJB administrative tasks are executed.

The `taskmanager-granularity` specified interval affects EJB timeouts. EJB administrative tasks associated with timeouts depend on when the task manager runs, and a factor of 60 for EJB tasks. Thus, if the `taskmanager-granularity` is changed from the default, the value specified for EJB timeouts will have a corresponding change in granularity.

> **See Also:** "Setting the OC4J Task Manager Granularity in server.xml" on page 6-23

Configuring Parameters for CMP Entity Beans

This section covers parameters for entity beans using CMP. These parameters are specified in the `orion-ejb-jar.xml` configuration file.

Table 6–7 lists the entity bean CMP specific parameters.

Table 6–8 describes the supported `locking-mode` parameter values.

This section also covers the following CMP topics:

- Configuring Lazy-loading on CMP Entity Bean Finder Methods
- Setting The CMP Define Column Type Option

Table 6–7 CMP Entity Bean Performance Parameters

Parameter	Description
`call-timeout`	For a description, see Table 6–6
`delay-updates-until-commit`	This boolean parameter, when `true`, specifies that sync and persistence only occur at the end of a transaction. If `false`, sync and persistence occur after every EJB method invocation, except `ejbRemove()` and the finder methods. Default Value: `true`
`do-select-before-insert`	If `false`, you avoid executing a select before an insert. The extra select normally checks to see if the entity already exists to avoid duplicates before doing the insert. If a unique key constraint is defined for the entity, then we recommend setting this to `false`. If there is no unique key constraint, setting this to `false` leads to not detecting a duplicate insert. To prevent duplicate inserts in this case, leave it set to `true`. For performance, Oracle recommends setting this to `false` to avoid the extra select before insert. Default Value: `true`
`exclusive-write-access`	Default is `false` for beans with `locking-mode=optimistic` or `locking-mode=pessimistic` and `true` for `locking-mode=read-only`. The `exclusive-write-access` is forced to `false` if locking is pessimistic or optimistic, and is not used with EJB clustering. The `exclusive-write-access` can be false with read-only locking, but read-only won't have any performance impact if `exclusive-write-access=false`, since ejbStores are already skipped when no fields have been changed. To see a performance advantage and avoid doing ejbLoads for read-only beans, you must also set exclusive-write-access=true.
`findByPrimaryKey-lazy-loading`	Turns on lazy loading in the `findByPrimaryKey` method. For entity bean finder methods, lazy loading can cause the select method to be invoked more than once. Default Value: `false`

Table 6–7 (Cont.) CMP Entity Bean Performance Parameters

Parameter	Description
`isolation`	If your database is already configured with the isolation mode you want for your transactions, you'll get better performance if you don't explicitly set the isolation mode attribute in the `orion-ejb-jar.xml file`. Omitting the isolation setting means to use the database default setting, and extra processing will not be done to explicitly set isolation levels in your transactions. See Table 6–9 for a description of `isolation` options and how they relate to locking modes. Default Value: When omitted, use the database default setting
`lazy-loading`	Specifies lazy loading on the finder-method element. Specifying this value to `true` turns on lazy loading for a custom finder method. See "Configuring Lazy-loading on CMP Entity Bean Finder Methods" on page 6-47 for more information. Default Value: `false`
`locking-mode`	The locking modes, specified with the `locking-mode` parameter, manage concurrency and configure when to block to manage resource contention or when to execute in parallel. See Table 6–8 for a description of `locking-mode`. See Table 6–9 for a description of `isolation` options and how they relate to locking modes. Default Value: `optimistic`
`max-instances`	See Table 6–6
`max-tx-retries`	See Table 6–6
`min-instances`	See Table 6–6
`pool-cache-timeout`	This parameter specifies how long to keep CMP Entity Beans cached in the pool. If you specify a `pool-cache-timeout`, then at every `pool-cache-timeout` interval, all beans in the pool of the corresponding bean type, are removed. If the value specified is 0 or negative, then the `pool-cache-timeout` is disabled and beans are not removed from the pool. In some cases it may help performance to disable `pool-cache-timeout`, or to set the `pool-cache-timeout` to a large value to avoid removing beans from the pool. Note: if `min-instances is > 0`, the `min-instances` number of instances are kept in the pool after the pool cache timeout (that is, they are not deleted). Note: if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this causes the `pool-cache-timeout` to be calculated based on the new taskmanager-granularity. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for details. Default Value: 60 (seconds)

Table 6–7 (Cont.) CMP Entity Bean Performance Parameters

Parameter	Description
`prefetch-size`	The `finder-method` element includes the `prefetch-size` attribute that specifies how many rows to prefetch into the client while a result set is being populated during a query. Using `prefetch-size` can reduce round trips to the database by fetching multiple rows of data each time data is fetched (the extra data is stored in client-side buffers for later access by the client). Increasing the value for the `prefetch-size` increases the memory needs for an application. It may be useful to increase the value from the default for finder-methods that fetch a lot of data, such as findAll on large tables, or custom finder-methods that retrieve many rows of data. You can see the affect of changing the `prefetch-size` in an application by looking at the finder-method avg time metric to see how much time it takes for the query, and how this affects the total response time for the application. The number of rows to prefetch can be set as desired using `prefetch-size`, however, for most applications using the default value, 10, is recommended. See Also: *Oracle9i JDBC Developer's Guide and Reference* for more information on using prefetch with a JDBC driver. Default Value: 10
`update-changed-fields-only`	Specifies whether the container updates only modified fields or all fields to persistence storage for CMP entity beans when `ejbStore` is invoked. When the value is set to `false`, this performs container updates to all fields to persistence storage, when `ejbStore` is invoked. When set to `false`, the container includes all fields in updates, so applications can take advantage of SQL statement caching. Default Value: `true`
`validity-timeout`	The `validity-timeout` is only used when `exclusive-write-access=true` and `locking-mode=read-only.` The validity timeout is the maximum time in milliseconds that an entity is valid in the cache (before being reloaded). We recommend that if the data is never being modified externally (and therefore you've set `exclusive-write-access=true`), that you can set this to 0 or -1, to disable this option, since the data in the cache will always be valid for read-only EJBs that are never modified externally. If the EJB is generally not modified externally, so you're using `exclusive-write-access=true`, yet occasionally the table is updated so you need to update the cache occasionally, then set this to a value corresponding to the interval you think the data may be changing externally.

Table 6–8 CMP Entity Bean Locking-Mode Values

Locking Mode Value	Description
`optimistic`	Multiple users can execute the entity bean in parallel. The optimistic locking mode does not monitor resource contention; thus, the burden of the data consistency is placed on the database isolation modes. This is the default value for `locking-mode`.
`pessimistic`	Manages resource contention and does not allow parallel execution. Only one user at a time is allowed to execute the entity bean. Pessimistic locking uses "`SELECT....FOR UPDATE`" to serialize access in the database.
`read-only`	Multiple users can execute the entity bean in parallel. The container does not allow any updates to the bean's state.

The `locking-mode`, along with `isolation`, assures database consistency for EJB entity beans using CMP. Table 6–9 shows the common `locking-mode` and `isolation` combinations. The different combinations have both functional and performance implications, but often the functional requirements for data consistency will lead to selecting a mode, even when it may be at the expense of performance.

Table 6–9 CMP Entity Bean Locking-Mode and Isolation Relationships

Locking-mode	Isolation	When to Use
`pessimistic`	`committed`	If data consistency must be guaranteed, and frequent concurrent updates to the same rows are expected.
`pessimistic`	`serializable`	We recommend that this combination not be used.
`optimistic`	`committed`	If concurrent reads and updates to the same rows with read-committed semantics is sufficient.
`optimistic`	`serializable`	If data consistency must be guaranteed, but infrequent concurrent updates to the same rows are expected.
`read-only`	`committed`	If repeatable read is not required.
`read-only`	`serializable`	If repeatable read is required.

In Table 6–9 the `isolation` setting refers to either the transaction `isolation` attribute setting, if explicitly set, or to the database isolation level (if the transaction `isolation` attribute is not set). Also, although `locking-mode` and transaction isolation levels are set as attributes of a CMP bean, the isolation level that will be in effect for the transaction is the isolation level of the first entity bean used in the transaction. Therefore it is best to set all beans in the same transaction to the same isolation level.

In general, optimistic locking with committed isolation will be faster since it allows for more concurrency, but it may not meet your needs for data consistency. Pessimistic locking with committed isolation, and optimistic locking with serializable isolation will be slower, but will guarantee data consistency on updates.

Defining a bean as read-only will assure that no updates are allowed to the bean. The performance will be similar to a bean which may not be defined as read-only, and yet is never used to do inserts, updates, or deletes (that is, only the methods which read are called). This is because if no fields are modified in a bean that is not defined with read-only locking, it is already optimized to not do an ejbStore. To see a performance advantage and avoid doing ejbLoads for read-only beans, you must also set `exclusive-write-access=true`.

Configuring Lazy-loading on CMP Entity Bean Finder Methods

Using CMP Entity Beans, each finder method retrieves one or more objects. In the default scenario, with `lazy-loading` set to `false`, no lazy-loading, each finder method causes a single SQL select statement to be executed against the database. For a CMP bean, one or more objects are retrieved with all of their CMP fields. So, for example, if you implement an `ejbFindAllEmployees` method, this finder retrieves all employee objects with all of the CMP fields in each employee object.

With `lazy-loading` set to `true`, only the primary keys of the objects retrieved within the finder are returned. Then, only when you access the object within your implementation, the OC4J container uploads the actual object based on the primary key. For example, with the `ejbFindAllEmployees` finder method, when `lazy-loading` is `true`, all of the employee primary keys are returned in a Collection. Then, each time you access one of the employees in the Collection, OC4J uses the primary key to retrieve the single employee object from the database.

The `lazy-loading` value should be set based on the performance considerations for your application. To determine whether `lazy-loading` should be set to `true` or `false`, lazy-loading is on or off, consider the following guidelines:

- If you use most of the retrieved objects, then you should set the `lazy-loading` option to `false` (use the default value).
- If you set `lazy-loading` to `true`, the first time an object is accessed within a transaction another select statement is executed, which results in a round-trip between the container and the database. If you only access a limited set of the

retrieved or found objects, or are doing a find only to verify existence, setting `lazy-loading` to `true` may improve performance.

- You may want to enable `lazy-loading`, set the value to `true`, if the finder method returns many rows with lots of data. With large data sets where the finder method does not return quickly, it may be better to set `lazy-loading` to `true`, enable lazy loading, so that the finder method returns quickly. After this, the application accesses rows as needed and the initial finder method return wait time can be reduced, which can improve application performance.

To turn on lazy-loading in the `findByPrimaryKey` method, set the `findByPrimaryKey-lazy-loading` attribute to `true`, as follows:

```
<entity-deployment ... findByPrimaryKey-lazy-loading="true" ... >
```

To turn on lazy-loading in any custom finder method, set the `lazy-loading` attribute to `true` in the `<finder-method>` element for that custom finder, as follows:

```
<finder-method ... lazy-loading="true" ...>
...
</finder-method>
```

Setting The CMP Define Column Type Option

Setting the `DefineColumnType` option to `true` in `server.xml` can improve performance for CMP entity beans, depending on the version of the JDBC driver.

> **See Also:** "Setting the OC4J Define Column Type Option" on page 6-10

Configuring Parameters for BMP Entity Beans

This section covers parameters that apply to entity beans using BMP. These parameters are specified in the `orion-ejb-jar.xml` configuration file.

Table 6–10 lists the entity bean BMP specific parameters.

Table 6–10 BMP Entity Bean Performance Parameters and Descriptions

Parameter	Description
`call-timeout`	See Table 6–6
`locking-mode`	The locking modes, specified with the `locking-mode` parameter, manage concurrency and configure when to block to manage resource contention or when to execute in parallel. BMP beans must use optimistic locking, which allows concurrent access to a bean, and the BMP bean is responsible for managing the database access and data consistency. It is up to the BMP bean to manage isolation as well, and therefore the isolation settings do not apply for BMP Default Value: `optimistic`
`max-instances`	See Table 6–6
`max-tx-retries`	See Table 6–6
`min-instances`	See Table 6–6
`pool-cache-timeout`	This parameter specifies how long to keep BMP Entity Beans cached in the pool. If you specify a `pool-cache-timeout`, then at every `pool-cache-timeout` interval, all beans in the pool of the corresponding bean type, are removed. If the value specified is 0 or negative, then the `pool-cache-timeout` is disabled and beans are not removed from the pool. In some cases it may help performance to disable `pool-cache-timeout`, or to set the `pool-cache-timeout` to a large value to avoid removing beans from the pool. Note: if `min-instances is > 0`, the `min-instances` number of instances are kept in the pool after the pool cache timeout (that is, they are not expired). Note: if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this causes the `pool-cache-timeout` to be calculated based on the new taskmanager-granularity. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for details. Default Value: 60 (seconds)

Configuring Parameters for Session Beans

This section covers the parameters that are specified in the `orion-ejb-jar.xml` configuration file and apply for session beans.

Table 6–11 lists the stateless session bean specific parameters.

Table 6–12 lists the stateful session bean specific parameters.

This section also covers the following topic:

- Configuring Stateful Session Bean Passivation
- Stateful Session Bean Passivation Performance Recommendations

Table 6–11 Stateless Session Bean Parameters

Parameter	Description
`call-timeout`	See Table 6–6
`pool-cache-timeout`	This parameter specifies how long to keep stateless session EJBs cached in the pool. For stateless session EJBs, if you specify a `pool-cache-timeout`, then at every `pool-cache-timeout` interval, all beans in the pool of the corresponding bean type, are removed. If the value specified is 0 or negative, then the `pool-cache-timeout` is disabled and beans are not removed from the pool. In some cases it may help performance to disable `pool-cache-timeout`, or to set the `pool-cache-timeout` to a large value to avoid removing beans from the pool. Note: if `min-instances is > 0`, the `min-instances` number of instances are kept in the pool after the pool cache timeout (that is, they are not expired). Note: if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this causes the `pool-cache-timeout` to be calculated based on the new taskmanager-granularity. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for details. Default Value: 60 (seconds)
`max-instances`	See Table 6–6
`max-tx-retries`	See Table 6–6
`min-instances`	See Table 6–6 Default Value: 0 (instances)

Table 6–12 Stateful Session Bean Parameters

Parameter	Description
`call-timeout`	See Table 6–6
`idletime`	Specifies the idle timeout for each Session Bean. When the bean has been inactive for the specified `idletime`, it is passivated. Default Value: 300 (seconds). Note: if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this causes the `idletime` to be calculated based on the new taskmanager-granularity. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for details. To disable, specify "never"
`max-instances`	The number of bean instances allowed in memory. When this value is reached, the container attempts to passivate the oldest bean instance from memory. If unsuccessful, the container waits the number of milliseconds set in the `call-timeout` attribute to see if a bean instance is removed from memory, either using passivation, the `remove()` method, or bean expiration, before a `TimeoutExpiredException` is thrown back to the client. To allow an unlimited number of bean instances, set `max-instances` to 0. To disable passivation due to reaching `max-instances`, set `max-instances` to 0. See Table 6–6
`max-instances-threshold`	Defines a threshold for how many active beans exist in relation to the `max-instances` attribute definition. Specify an integer that is translated as a percentage. For example, setting the `max-instances` to 100 and the `max-instances-threshold` to 90 (90%), specifies that when active bean instances reach past 90, passivation of beans occurs. The number of beans that are passivated after crossing this threshold is specified with the `passivate-count` parameter. Default Value: 90% To disable, specify "never"
`max-tx-retries`	Ssee Table 6–6
`memory-threshold`	Defines a threshold for how much used JVM memory is allowed before passivation should occur. Specify an integer that is translated as a percentage. When the threshold is reached, beans are passivated, even if their idle timeout has not expired. The number of beans that are passivated after crossing this threshold is specified with the `passivate-count` parameter. Default Value: 80% To disable, specify "never"

Table 6–12 (Cont.) Stateful Session Bean Parameters

Parameter	Description
`passivate-count`	This attribute is an integer that defines the number of beans to be passivated if any of the resource thresholds have been reached. Passivation of beans is performed using the least recently used algorithm. Default Value: one-third of the `max-instances` attribute (if `max-instances` is > 0). If `max-instances` is 0, `passivate-count` defaults to 0 (disabled). To disable `passivate-count`, set the value to 0 or to a negative number.
`resource-check-interval`	The container checks all resources at this time interval. At this time, if any of the thresholds have been reached, passivation occurs. Default Value: 180 seconds (3 minutes). To disable, specify "never"
`timeout`	Specifies the `timeout` for Stateful Session EJBs in seconds. If the bean has been inactive for the specified `timeout`, the bean is invalidated or removed. If the value is set to zero (0) or to a negative value, then the `timeout` is disabled. When a Stateful Session EJB is inactive, after the `timeout` expires, it is invalidated and a request for the bean returns `NoSuchObjectException` to the client. When the pool clean-up logic is invoked (by default every 30 seconds), the pool clean-up logic invalidates or removes the sessions that timed out, (sessions with expired `timeout` values). Adjust the `timeout` based on your applications use of Stateful Session EJBs. For example, if your application does not explicitly remove Stateful Session EJBs, and the application creates many Stateful Session EJBs, then you may want to lower the `timeout` value. If your application requires that a Stateful Session EJBs be available for longer than 1800 seconds, 30 minutes, then adjust the `timeout` accordingly. Note: if you change the default value of the `taskmanager-granularity` attribute in `server.xml`, this causes the `timeout` to be calculated based on the new taskmanager-granularity. See "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42 for details. Default Value: 1800 (seconds)

Configuring Stateful Session Bean Passivation

Passivation for a Stateful Session Bean (SFSB) is invoked based on any combination of the following criteria:

- The idle timeout expires for a bean instance. The idle timeout is specified with the `idletime` parameter.
- The container is determined to be out of resources, where a resource to be monitored is specified with the following parameters.
  - `memory-threshold`
  - `max-instances-threshold`

  Note: If you use either of these parameters for container resource control, then setting the `resource-check-interval`, and `passivate-count` parameters is mandatory.

- The number of bean instances allowed is reached as defined in the `max-instances` parameter in the `<session-deployment>` element in `orion-ejb-jar.xml` (see Table 6–12 for details).
- The OC4J instance terminates: the non passivated beans in memory are flushed to storage when the OC4J instance shuts down.

The attributes that control the Stateful Session Bean (SFSB) passivation management are configured in the `<session-deployment>` tag of the `orion-ejb-jar.xml` file for the deployed application.

Enabling passivation for the entire OC4J instance is configured at the container level in `server.xml` using the `<sfsb-config>` tag with the attribute `enable-passivation`. When `enable-passivation=false` this disables all the bean level settings for passivation and disables passivation at OC4J instance termination. When `enable-passivation=true` applications can control bean passivation and passivation management using the passivation control parameters (see Table 6–12 for details).

Passivation is enabled by default and each stateful session bean is configured to passivate when any SFSB's `idletime` expires, by default after 5 minutes, and when the OC4J instance terminates. By default, resource-based and max-instances based passivation is not enabled.

See Also:

- "Setting the OC4J Options for Stateful Session Bean Passivation in server.xml" on page 6-27
- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for information on EJB Lifecycle Issues

Stateful Session Bean Passivation Performance Recommendations

The Stateful Session Bean (SFSB) activation and passivation model is analogous to using swap space at the OS level – when certain operating characteristics are met, the in-memory state of qualified beans is flushed to disk, allowing more users to be served.

There is a performance overhead involved with passivation (which makes additional memory available). The overhead occurs when the state of the SFSB is written to disk, and when the SFSB is subsequently reused and the SFSB must be read from disk and activated. Therefore, if the configuration specified for the passivation parameters is "incorrect", this can cause significant passivation activity, and the "extra" passivation activity can degrade performance. Specifying passivation parameters with "incorrect" values can also use up disk space when a large amount of state is maintained in the SFSBs and when the beans are not expired (or do not expire for a very long time).

When your application is not affected by memory limitations, the best performance for SFSBs is achieved by disabling passivation completely, system wide, in `server.xml`, or by setting parameters for each individual bean so that SFSB passivation is rarely used.

If the OC4J instance has passivation enabled, it will always passivate active beans in memory at shutdown.

To turn off all other kinds of passivation for individual beans, use the following parameters with the following values (see Table 6–12 for details):

```
idletime=never
passivate-count=0
max-instances=0
max-instances-threshold=never
memory-threshold=never
resource-check-interval=never
```

When disabling passivation for an individual bean, note the following:

- If you explicitly set `passivate-count=0`, this also disables `memory-threshold` and `max-instances-threshold`.
- If you explicitly set `resource-check-interval=never`, this also disables `memory-threshold` and `max-instances-threshold.`
- You can passivate based on `max-instances` with or without setting a `max-instances-threshold`.

If you enable passivation to help control memory usage, you can improve performance by limiting the use of passivation (when possible). The following options are available to help control memory usage by SFSBs without requiring passivation:

- Using Timeouts: specify the minimum timeout for the SFSB that your application requirements allow (using the `timeout` attribute specified in `orion-ejb-jar.xml`, see Table 6–12 for details). When a SFSB expires due to timeout, it is removed and not passivated (if it reaches timeout before the idletime timeout and before other passivation criteria are reached).
- Using the `remove()` method: if you know in the application when you are done using a particular SFSB, then you should call the bean `remove()` method to release its memory rather than letting the bean timeout or be passivated.

The following are additional guidelines to help you decide if you need to use passivation:

1. Generally, if you do not reuse SFSBs quickly, then set the `timeout` and the `idletime` so the beans are removed without requiring passivation. To prevent passivation, set the timeout to be short and set the `idletime` to a long time, or to `never`, so that beans are not passivated before being removed (if you have sufficient memory to handle the load).

   For example, consider an application where you create 1000's of SFSBs within 5 minutes, and you expect most of these beans to be idle for at least 5 minutes after first use and subsequently reused within 30 minutes. The default timeout is 30 minutes and the default idletime is 5 minutes. Then, in this case, it would be good to either increase the `idletime` to 30 minutes or disable passivation based on `idletime`. This guideline helps avoid having 1000's of SFSB passivated to disk, which has a costly performance overhead (the guideline also assumes you will not run into memory limitations by making this configuration change).

2. Consider setting `max-instances`, `idletime`, or memory resource thresholds to limit the number of beans in memory if:

   - You cannot fit all the SFSBs your client load generates and needs over a period of time (the timeout period of time) in memory.
   - You do want to save the state, since you know you will typically reuse it.
   - You cannot reduce the timeout for the SFSBs to reduce how many are saved.

3. You can look at the metrics for the methods `create`, `ejbPassivate`, `ejbActivate`, and `ejbRemove` on the SFSB to see how many stateful beans are created and how much passivation is occurring.

4. Set `task-manager-granularity` to 1000 to get greater accuracy on tasks occurring near the timeout values for EJBs.

   See Also: "EJB Timeouts Using a Non-default taskmanager-granularity" on page 6-42

Configuring Parameters for Message Driven Beans (MDBs)

This section covers the EJB parameters specified in the `orion-ejb-jar.xml` configuration file that apply for Message Driven Beans (MDBs).

Table 6–13 lists the MDB specific parameters.

Table 6–13 Message Driven Bean orion-ejb-jar.xml Parameters

Parameter	Description
dequeue-retry-count	Specifies how many times the listener thread is to try to re-acquire the JMS session once a database failover has occurred. Setting the dequeue-retry-count can be useful when running with a RAC-enabled database cluster. Note: this parameter only applies to OJMS. Default Value: 0
`dequeue-retry-interval`	Specifies how often the listener thread is to try to re-acquire the JMS session once a database failover has occurred. Setting the `dequeue-retry-interval` can be useful when running against a RAC-enabled database cluster. Note: this parameter only applies to OJMS. Default Value: `60` (seconds)
`listener-threads`	If set to a value greater than 1, `listener-threads` enables multiple instances of the MDB to concurrently process messages from queues. Use `listener-threads=1` if the messages must be processed in order. See Also: "Using The listener-threads MDB Parameter" on page 6-58 for a detailed description of the `listener-threads` parameter and for limitations. Default Value: `1`
`transaction-timeout`	Specifies the maximum time taken for a transaction to finish before it is rolled back due to a timeout (this parameter only applies for an MDB that uses container-managed transactions). The MDB transaction timeout timer starts when the listener thread starts listening for a new message. Note: the `server.xml` timeout value, specified with `transaction-config timeout` does not apply to MDB operations. Set the `transaction-timeout` to a value that is greater than the longest expected transaction time. If the `transaction-timeout` is set too small, this can cause unnecessary rollback and retry overhead. When a timeout occurs, the MDB automatically does a rollback of the current transaction and the associated messages will be redelivered for retry. To check for transaction-timeouts, view the `application.log` for entries containing the following: `javax.transaction.SystemException(timed out)` Default Value: 86,400 seconds (1 day) When using OracleAS JMS, the transaction timeout cannot be altered from the default value.

See Also: "Java Message Service" in the *Oracle Application Server Containers for J2EE Services Guide* for information on using MDBs with OracleAS JMS and OJMS

Using The listener-threads MDB Parameter

Setting the `listener-threads` parameter for an MDB can improve performance when there are many concurrent users sending messages to an MDB's queue, or when the processing that occurs in the `onMessage` method is significant. For example, if the `onMessage` method contains code to call another EJB and the EJB processing can occur concurrently while processing other messages, then specifying a `listener-threads` value greater than one can improve performance. Depending on the underlying JMS provider and the specific MDB, some applications may see significant performance improvements by increasing the number of listener threads.

When the `listener-threads` parameter is specified for an MDB, the OC4J runtime creates the specified number of threads to service messages for the MDB and specifies the degree of parallelism for the MDB. The listener threads are created when the MDB starts at OC4J startup.

For example, if a queue contains 100 messages, and the `listener-threads` parameter is set to the default value, 1, then only one MDB listener-thread processes the messages, in a serial fashion. If the `listener-threads` parameter is set to 5, there can be a maximum of 5 MDB instances that take messages from the queue, and process the messages in parallel. The total time required to complete the processing for 100 messages can be shortened since OC4J uses 5 MDB threads to dequeue and process the messages.

In a multiuser test, with 10 users, where `listener-threads` is set to 5, compared to using the default value, 1, end-to-end performance improved by a factor of 2. This test involved a Servlet sending a message to an Oracle JMS queue, and then the MDB receiving the message from the queue and sending a reply to a reply queue.

In another test, using OracleAS JMS with `listener-threads` set to 5, compared to the default value 1, throughput increased by 27%.

Note: Using the `listener-threads` parameter, any performance improvement depends on the application and on the number of threads specified. Specifying a value that is too large may cause performance to degrade due to resource contention.

Notes for using `listener-threads`:

1. The number of `listener-threads` is included in the total global thread pool thread count specified using the `max` thread pool parameter. Consider that the `listener-threads` number of threads will be dedicated to MDB processing; therefore, you need to allocate this number, plus sufficient additional threads in the global thread pool to handle other OC4J processing.

   See Also: "Setting the OC4J Global Thread Pool Attributes in server.xml" on page 6-24

2. When using OJMS, the number of `listener-threads` is also the number of dedicated database connections that the MDB uses. So, the number of `listener-threads` must be included in the total datasource specified `max-connections` count.

   See Also: "Setting the Maximum Open Connections in Data Sources" on page 6-15

3. The `listener-threads` parameter is not supported for topics. Thus, topics can have at most one thread processing in an MDB.

4. Using `listener-threads` with a value greater than 1, messages are still removed from a queue serially, but the order of processing the messages cannot be guaranteed since the MDB is processing the messages with multiple threads. Use `listener-threads=1`, the default value, when the order of message processing is important. This assumes that the MDB is solely responsible for receiving messages from the queue.

Using Performance Metrics for MDB Messages

When MDBs use OracleAS JMS as a message provider, DMS message related metrics are available from the Oracle Application Server performance monitoring tools.

For example, the OracleAS JMS `JMSStoreStats` metric table includes information for a destination corresponding to a queue that an MDB uses:

```
destination.value:        name
messageDequeued.count:    x ops
messageEnqueued.count:    x ops
messageCount.value:  n
```

These metrics show the destination name, the total messages enqueued, the total number of messages dequeued, and the total number currently in the queue.

Note: When monitoring a JMS destination, other applications besides the MDB may access the destination. Thus, when testing your application's performance using the metrics, make sure that you know whether your application is responsible for message activity reported in the metrics.

See Also: "OC4J JMS Metrics" on page A-19

Setting up JMS Connections in MDB ejbCreate or onMessage Methods

An MDB is stateless and contains no specific client state across invocations. However, for non-client related state, an MDB instance can contain some state across the handling of client messages. For example, state can be maintained for a JMS API connection object. In addition, other state information that you may want to cache across `onMessage` invocations, such as a reference to an EJB, can be initialized in `ejbCreate` method and cached to optimize MDB performance. Depending on the application and the message provider, you may be able to improve performance by selecting when JMS connections, JMS sessions, and other objects are initialized, either in the MDB `ejbCreate` method or in `onMessage`.

Table 6–14 summarizes some performance recommendations for selecting when to create JMS connections and JMS sessions using OracleAS JMS and Oracle JMS (OJMS).

Table 6–14 JMS Performance Recommendations With ejbCreate and onMessage

JMS Provider	Performance Recommendation
OracleAS JMS	To optimize performance initialize the JMS connection and session once in the MDBs `ejbCreate()` method, and use repeatedly across `onMessage` invocations.
Oracle JMS	You cannot cache JMS sessions to the database across `onMessage` invocations. So, for any queues or Topics used in an MDB, you should set up the Queue or Topic Connection, Session, and Sender in the `onMessage` method of the MDB, and close them at the end of `onMessage` method. Do not create these objects in the `ejbCreate()` method of the MDB and then leave them open indefinitely, since these objects open and close logical connections to the database. The overhead of opening and closing connections and sessions in the `onMessage` method should not be significant, and the physical connections can then be reused.

Improving Web Services Performance in Oracle Application Server

In Oracle Application Server, the tuning guidelines for J2EE applications in general apply to Web Services. Specifically, because Web Services use Java Servlets for entry points, the guidelines for improving Servlet Performance apply to Oracle Application Server Web Services. In addition, when a Web Service is implemented as an EJB, the performance guidelines for EJBs apply.

This section covers the following topics:

- Avoiding Web Services Initial Request Delay
- Using Web Services Typed Requests
- Tuning The Web Services Stateful Session Timeout

> **See Also:** "Improving Servlet Performance in Oracle Application Server" on page 6-28

Avoiding Web Services Initial Request Delay

Oracle Application Server Web Services may experience an initial request delay due to the work required to validate data types and to generate server skeleton code. As a result, the initial Web Service request takes substantially longer than subsequent requests. In our tests, we see the first test taking 5 to 10 times as long as subsequent requests. The delay is increased when Java Beans are used to represent complex parameter and result sets.

To prevent this delay, send a request to Web Services on the system when the system is restarted or when the application is redeployed. You can also produce a script to send the initial Web Service request.

Using Web Services Typed Requests

There is a performance overhead associated with using Web Services untyped requests. When possible, develop clients that use typed requests as un-typed requests will take more time on the first request when the SOAP Mapping registry is created for the operation types.

> **See Also:** Chapter 12, "Advanced Topics for Web Services" in the *Oracle Application Server Web Services Developer's Guide* for more information.

Tuning The Web Services Stateful Session Timeout

When using Stateful Session based Web Services, tuning the `session-timeout` property for session-scoped stateful applications can provide performance benefits. The HTTP session timeout is specified in the `web.xml` configuration file as the `<session-timeout>` sub-element of the `<session-config>` element.

> **See Also:** Chapter 2, "Servlet Development" in *Oracle Application Server Containers for J2EE Servlet Developer's Guide*

Improving BC4J Performance in Oracle Application Server

This section contains tips for improving the maintainability, scalability, and performance of your Oracle Business Components for Java (BC4J) applications.

Choose the Right Deployment Configuration

Your application will have the best performance and scalability if you deploy your business components to the web module with your client. Unless you have strong reasons (such as wanting to use distributed transactions or EJB security features), we recommend web module deployment of business components over EJB deployment.

Note that both web module deployment and EJB deployment are fully J2EE-compliant, and the BC4J framework makes it easy to switch between them. You can test your application in both modes to see which gives you the best performance.

Use Application Module Pooling for Scalability

A client can use application module instances from a pool, called application module pooling. This offers these advantages:

- It reduces the amount of time to obtain server-side resources
- It allows a small number of instances to serve a much larger number of requests
- It addresses the requirements of web applications that must handle thousands of incoming requests
- It lets you preserve session state and provides failover support

For example, in the case of a web application, you may have 1,000 users but you know that only 100 will be using a certain application module at one time. So you use an application module pool. When a client needs an application module instance, it takes a free one from the pool and releases it to the pool after either committing or rolling back the transaction. Because the instance is precreated, end users are saved the time it takes to instantiate the application module when they want to perform a task. Typically, web-based JSP clients use pools. If you want to make sure that the application module pool has a maximum of 100 application module instances, you can customize the default application module pool.

If your client needs to keep track of application module state, we recommend using stateful mode. In a stateful JSP application, the client does not reserve the application module instance, making it available to other clients if the number of application modules exceeds the recycle threshold. State is, instead, preserved in one of two ways: The application module pool returns a client's original application module if the application module has not been recycled, and the pool persists the state of recycled application modules in the database to be available to clients that request them later.

When you release an application module at the end of a user's session, be sure to use stateless (rather than stateful or reserved) release mode. This frees up database space and allows the pool to recycle the application module immediately.

Perform Global Framework Component Customization Using Custom Subclasses

Particularly in large organizations, you may want specific functionality shared by all components of a particular type--for example, by all view objects. An architect can create a thin layer of classes such as `MyOrgViewObjectImpl` that implement the desired behavior. Individual developers can extend `MyOrgViewObjectImpl` instead of `ViewObjectImpl`, and you can use the "substitutes" feature to extend `MyOrgViewObjectImpl` in legacy code.

Use SQL-Only and Forward-Only View Objects when Possible

Basing a view object on an entity object allows you to use the view object to insert, update, and delete data, and helps keep view objects based on the same data synchronized. However, if your view object is only going to be used for read-only queries, and there is no chance that the data being queried in this view object will have pending changes made through another view object in the same application module, you should use a SQL-only view object that has no underlying entities. This will give you improved performance, since rows do not need to be added to an entity cache.

If you are scrolling through data in one direction, such as formatting data for a web page, or for batch operations that proceed linearly, you can use a forward-only view object. Forward-only mode prevents data from entering the view cache. Using forward only mode can save memory resources and time, because only one view row is in memory at a time. Note that if the view object is based on one or more entity objects, the data does pass to the entity cache in the normal manner, but no rows are added to the view cache.

Do Not Let Your Application Modules Get Too Large

A root application module should correspond to one task--anything that you would include in a single database transaction. Do not put more view objects or view links than you will need for a particular task in a single application module.

In addition, consider deferring the creation of view links by creating them dynamically with `createViewLink()`. If you include all view links at design time, the business logic tier will automatically execute queries for all detail view objects when your client navigates through a master view object. Deferring view link creation will prevent the business logic tier from executing queries for detail view objects that you do not yet need.

For example, for a form in which detail rows are displayed only on request (rather than automatically), including a view link at design time would force the business logic tier to automatically execute a query that might well be unnecessary. To prevent this, you should create a view link dynamically when the detail rows are requested. By contrast, for a form in which detail rows are displayed as soon as a master is selected, you should use a view link created at design time to avoid the runtime overhead of calling `createViewLink()`.

Use the Right Failover Mode

By default, the application module pool supports failover, which saves an application module's state to the database as soon as the application module is checked into the pool. If the business logic tier or the database becomes inoperable in mid-transaction (due to a power failure or system malfunction, for example), the client will be able to instantiate a new application module with the same state as the lost one, and no work will be lost.

However, some applications do not require this high level of reliability. If you're not worried about loss of work due to server problems, you may want to disable failover. When failover is disabled, the application module's state exists only in memory until it is committed to the database (at which point the application module's state is discarded) or recycled (at which point the state is saved so that the client can retrieve it). By not saving the application module state every time the application module is checked in, failover-disabled mode can improve performance.

Use View Row Spillover to Lower the Memory to Cache a Large Number of Rows

While the business logic tier is running, it stores view rows in a cache in memory (the Java heap). When the business logic tier needs to store many rows at once, you need to make sure it doesn't run out of memory. To do so, you can specify that when the number of rows reaches a certain size, the rows "overflow" to your database to be stored on disk. This feature is called view row spillover. If your application needs to work with a large query result, view row spillover can help the cache operate more efficiently.

Choose the Right Style of Bind Parameters

Oracle-style bind parameters (:1, :2, and so on) are more performant than JDBC-style bind parameters.

There are only two reasons to use JDBC-style bind parameters:

- Use JDBC-style bind parameters if you may use a non-Oracle JDBC driver.
- Use JDBC-style bind parameters if you have more than one occurrence of the same parameter in the `WHERE` clause.

Implement Query Conditions at Design Time if Possible

You should include any portion of your query condition that you know in advance in the `WHERE` clause field in the View Object wizard. Only use `setWhereClause()` for genuinely dynamic query conditions.

Even if your query conditions are genuinely dynamic, you may be able to use parametrized queries instead of `setWhereClause()`. For example, if your view object needs to execute a query with the `WHERE` clause `EMPLOYEE_ID=<x>` for various values of x, use a parametrized `WHERE` clause such as `EMPLOYEE_ID=:1`. This is more efficient than repeatedly calling `setWhereClause()`.

Use the Right JDBC Fetch Size

The default JDBC fetch size is optimized to provide the best tradeoff between memory usage and network usage for many applications. However, if network performance is a more serious concern than memory, consider raising the JDBC fetch size.

Turn off Event Listening in View Objects used in Batch Processes

In non-interactive, batch processes, there is no reason for view objects to listen for entity object events. Use `ViewObject.setListenToEntityEvents(false)` on such view objects to eliminate the performance overhead of event listening.

Improving JAAS (JAZN) Performance in Oracle Application Server

The Java Authentication and Authorization Service (JAAS) is a package that supports user and role-based authorization, authentication, and delegation. Part of JAAS is an implementation of the standard Pluggable Authentication Module (PAM) framework in Java, which supports the separation of an application from its underlying authentication technologies. Oracle Application Server provides an integrated JAAS implementation with OC4J called JAZN and provides a login module, out of the box, that supports several common forms of authentication.

When performing authentication and authorization operations, JAZN accesses a repository of data that defines users, roles, permissions, and related information. The characteristics of the repository are important to the performance and scalability of applications that use JAZN.

Oracle Application Server JAZN provides two types of repository provider for use with OC4J:

- XML provider – The XML provider stores repository information in an XML file
- LDAP provider – The LDAP provider stores repository information in the Oracle Internet Directory, which is accessed using the Lightweight Directory Access Protocol (LDAP)

This section covers the following topics:

- Improving JAZN Performance With an XML Provider
- Improving JAZN Performance With an LDAP Provider (Oracle Internet Directory)
- Configuring JAZN Providers
- JAZN Performance Recommendations

Improving JAZN Performance With an XML Provider

When OC4J with JAZN is configured to use the XML provider, JAZN loads the entire XML file into a data structure in memory for fast access. In terms of performance, this process incurs a small start-up cost, but if the file is not too large and the data in the file can be retained in physical memory, data access will be very efficient and JAZN operations should incur little overhead.

> **See Also:** *Oracle Application Server Containers for J2EE Security Guide*

Improving JAZN Performance With an LDAP Provider (Oracle Internet Directory)

When OC4J applications using JAZN are configured to use an LDAP provider, the LDAP repository is queried for data on demand. In this case, a single operation may involve multiple accesses to a remote directory, and the overhead for JAZN protection can become significant. Such overhead can be even greater if secure communications are required between OC4J and the repository which typically requires using SSL. When JAZN is configured to communicate with the LDAP repository using SSL, the performance issues of the SSL protocol should be considered.

There are several configuration choices to make when you set up SSL between OC4J, and LDAP (Oracle Internet Directory). SSL can be configured to use encryption only, or encryption plus client or server authentication.

To alleviate the costs of communicating with an LDAP repository, OC4J JAZN provides caches, including the following three separate caches:

- The Policy Cache: stores grantees and permissions
- The Realm Cache: stores realms, users and roles
- Session cache: stores users and roles in an HTTP session object

The JAZN-LDAP caches are implemented as a single, in-memory hashtable. Objects in the cache are expired based on a configurable timeout value. A daemon thread runs periodically, at the timeout interval, to clean up expired objects in the cache. Each of the three caches can be enabled or disabled, and the initial capacity, load factor, initial cache purge delay, and cache purge timeout value can all be specified.

By default, the JAZN LDAP Provider is configured to use caching. Caching greatly improves the efficiency of using JAZN with an LDAP-based repository. Our experiments have shown the default values of cache configuration often work well, but you may need to test these values to determine how your application performs using JAZN.

See Also:

- *Oracle Application Server Containers for J2EE Security Guide*
- *Oracle Internet Directory Administrator's Guide*

Configuring JAZN Providers

Oracle Application Server OC4J provides an integrated JAAS implementation with OC4J. To configure the JAAS provider, you use `jazn.xml` to determine if the provider is LDAP-based, uses Oracle Internet Directory as the data store, or XML based.

The file `jazn.xml` is the configuration file for both the XML-based and LDAP-based JAAS providers. The JAAS Provider must locate a valid `jazn.xml` file before it can begin running.

When the JAAS provider starts up, it searches for `jazn.xml` in order through the directories specified by:

1. `oracle.security.jazn.config` (system property)
2. `java.security.auth.policy` (system property)
3. `$J2EE_HOME/config` (`$J2EE_HOME` is specified by the system property `oracle.j2ee.home`)
4. `$ORACLE_HOME/j2ee/home/config` (`$ORACLE_HOME` is specified by the system property `oracle.home`)
5. `. ./config`

The JAAS provider stops searching after locating a `jazn.xml` file. If no file is found, you receive the error message "JAZN has not been properly configured."

You can also use the `<jazn>` tag to configure the JAAS Provider. The `<jazn>` tag can appear in any of the following locations:

- The application's `orion-application.xml`
- The global `application.xml`
- `jazn.xml`

See Also: *Oracle Application Server Containers for J2EE Security Guide*

Configuring Session Timeout in web.xml

The JAZN session cache can only be used by HTTP clients that have cookies enabled. Objects in this cache are held for the duration of an HTTP session. The HTTP session timeout is specified in the `web.xml` configuration file as the `<session-timeout>` sub-element `<session-config>` element.

> **See Also:** *Oracle Application Server Containers for J2EE Servlet Developer's Guide*

JAZN Performance Recommendations

The following recommendations should help you to meet the performance requirements for applications that use JAZN for authentication and authorization:

1. If the JAZN XML file-based repository is sufficient for your needs, it is likely to provide the best performance.
2. If an LDAP repository is required, for management, usability, or scalability reasons, use the JAZN-LDAP caches. Configure the cache parameters as needed to improve performance.
3. If an LDAP repository is required, and if secure communications are needed between the LDAP repository and OC4J, configure the system to use only the level of security required. For example, use encryption only if that is sufficient.

Using Multiple OC4Js, Limiting Connections and Load Balancing

This section outlines areas that allow you to improve performance by setting the number of processes in an OC4J Instance, by directing requests to different OC4J Instances, and by limiting the number of requests sent to an OC4J Instance. These techniques spread the J2EE application load and the incoming requests among multiple OC4J processes which generally results in higher throughput and shorter response time. In addition, multiple OC4J processes are needed for load-balancing, high availability, and failover.

This section provides links to other Oracle Application Server documents and sections in this guide that show you how to configure and use multiple OC4Js.

> **Note:** The replication features that provide for failover with Web sessions and for stateful session EJBs have a performance overhead; only use these features when failover features are needed.

This section covers the following topics:

- Configuring Multiple OC4J Processes
- Load Balancing Applications
- Limiting Connections
- Controlling Replication With Multiple OC4Js

Configuring Multiple OC4J Processes

This section covers the following:

- Overview of Types of OC4J Configurations
- Determining the Number of OC4J Processes
- Partitioning Applications into Different OC4J Instances
- Configuring Multiple OC4J Processes Using Application Server Control

Overview of Types of OC4J Configurations

Oracle Application Server supports different types of installations and configurations, where you can run multiple OC4Js, including the following:

- A standalone Oracle Application Server Instance with multiple OC4J Instances (each OC4J Instance may include multiple OC4J processes).
- Oracle Application Server Clusters, managed, where a collection of application server instances runs with identical configurations and application deployments.
- Oracle Application Server Clusters, non-managed, where the administrator manually configures each instance within a cluster.
- A single or multiple hosts running standalone OC4J.

Determining the Number of OC4J Processes

Determining the optimal ratio of OC4J processes to available CPUs is dependent on the characteristics of the applications you run, the OC4J configuration, the hardware configuration, and the type and number of expected incoming requests. In general, for multi-CPU configurations with greater than two processors, you should consider configuring multiple OC4J processes. For example, on a recent test of a J2EE application, a single OC4J process was sufficient to use most of the CPU resources on a 2 processor system. Adding additional OC4J processes will not help improve performance on this system. However, on a six processor system, a single OC4J process uses only 70% of the CPU resources. Since additional CPU resources are available on this system, adding a second OC4J process should improve performance.

Adding processes beyond the available resources of the system will not improve performance. For example, if one OC4J process is sufficient to saturate the CPU resources of a system, adding additional processes is not likely to improve performance and may, in fact, degrade it. A good starting point is to configure one OC4J process for every 3-4 CPUs and measure the improvement from adding additional processes.

See Also:

- *Oracle Application Server 10g High Availability Guide*
- *Oracle Application Server Containers for J2EE User's Guide*

Partitioning Applications into Different OC4J Instances

If your Oracle Application Server has many different applications deployed, each of which has different requirements, you may want to configure different OC4J Instances to service the different applications (and OC4J Instances may be configured with different numbers of OC4J processes).

To deploy applications to different OC4J Instances, perform the following steps:

1. Create the multiple OC4J Instances.
2. Use the Deploy Application Wizard, by selecting the **Deploy Ear File** button, on each Instance, and deploy the appropriate application and specify a unique URL mapping for each of the applications.

After deploying the applications to different OC4J Instances, you can monitor the performance to see if overall throughput increases, or the response time decreases.

See Also: *Oracle Application Server Containers for J2EE Servlet Developer's Guide*

Configuring Multiple OC4J Processes Using Application Server Control

Using Application Server Control you can specify the number of processes in an OC4J Instance from the Server Properties page. This page is available by selecting the Administration link from an OC4J Instance page.

See Also: *Oracle Application Server 10g High Availability Guide*

Load Balancing Applications

OC4J provides load-balancing features for web-based applications with HTTP clients and for EJB applications accessed by remote Java EJB clients.

This section covers the following topics:

- Web Application Load Balancing
- EJB Application Load Balancing

Web Application Load Balancing

In an Oracle Application Server environment, the Oracle HTTP Server uses `mod_oc4j` to load balance requests between the available OC4J processes. In this environment you can select `mod_oc4j` configuration options to choose the appropriate `mod_oc4j` load balancing policies to improve performance.

> **See Also:** "Setting mod_oc4j Load Balancing Policies" on page 5-22

EJB Application Load Balancing

After an EJB application is deployed to multiple OC4Js, an EJB client-side application can load balance its requests across the available OC4Js. To use load balancing, the client-side application configures the JNDI properties to use load balancing.

There are three ways that the EJB client-side application can set the JNDI properties, including:

- Setting the properties in the environment passed to the InitialContext
- Setting the properties in the `jndi.properties` file
- Setting the JVM system parameters on the client-side OC4J

This section shows the EJB client-side properties that are specified in the `jndi.properties` file. This section shows the load balancing related properties, but does not include all the available properties.

See Also:

- "Setting JNDI Properties", in Chapter 2 of *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*
- Chapter 10, in *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for more information on load-balancing EJBs

Setting the JNDI java.naming.factory.initial Property

The `java.naming.factory.initial` property specifies the initial context factory to use.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

Setting the JNDI java.naming.provider.url Property

Oracle Process Manager and Notification Server (OPMN) dynamically sets all ports, including the RMI port, when each OC4J instance starts.

Using the `java.naming.provider.url` property in the EJB client-side JNDI properties, the client-side OC4J retrieves a list of the available dynamic ports for the OC4J instance, and if the OC4J instance is part of a cluster, a list of all the available dynamic ports for that instance across the cluster. If the list includes more than one port, the EJB client-side code randomly picks one port from the list to send your requests to. All EJB lookups using that InitalContext will go to the selected host.

Use the following syntax for setting the URL, including the `opmn:ormi:` prefix for the `java.naming.provider.url` property:

```
opmn:ormi://opmn_host:opmn_port:oc4j_instance/application-name
```

The OPMN host name, *`opmn_host`*, and port number, *`opmn_port`*, is retrieved from the `$ORACLE_HOME/opmn/conf/opmn.xml` file.

In most cases, OPMN is located on the same machine as the OC4J instance. However, you must specify the host name in case it is located on another machine. The OPMN port number is optional; if excluded, the default is port 6003. The OPMN port is specified in the file `$ORACLE_HOME/opmn/conf/opmn.xml`.

Setting the JNDI java.naming.provider.url Property in Standalone OC4J

For standalone OC4J, specify the `java.naming.url` property using a comma separated list of URLs including the `ormi:` prefix and the hosts where OC4J runs. This load-balances EJB client get InitialContext requests randomly across the hosts and OC4J processes specified in the comma separated list. All EJB lookups using that InitalContext will go to the selected host.

The syntax for specifying each URL for a host is as follows:

```
ormi://hostname:ormi_port/application-name
```

The ORMI port, *`ormi_port`*, can be omitted if the port is the default ORMI port number (23791).

For example, to load balance to `my_ejb_app` that is running on host1, host2, and host3, set the property `java.naming.provider.url` as follows:

```
java.naming.provider.url=ormi://host1:23791/my_ejb_app,ormi://host2:23792/
my_ejb_app,ormi://host3:23791/my_ejb_app
```

Setting the JNDI java.naming.security.principal Property

Setting the `java.naming.security.principal` property specifies the username.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

Setting the JNDI java.naming.security.credentials Property

Setting the `java.naming.security.credentials` property specifies the password.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

Setting the OC4J Dedicated RMI Context Option for Remote EJB Clients

When you set the property `dedicated.rmicontext=true`, then each initial context lookup receives its own InitialContext instead of a shared context. This option is only needed if an EJB client is doing multiple initial context lookups within the same JVM and you want to use load balancing.

When the property `dedicated.rmicontext` is `false`, OC4J load balances only on the first get initial context call. This `dedicated.rmicontext` property is set to `false` by default.

Limiting Connections

This section covers the following topics:

- Limiting Web Connections
- Limiting Remote EJB Client Connections
- Limiting HTTP Connections with Standalone OC4J

Limiting Web Connections

You can improve J2EE application performance by limiting the number of active HTTP concurrent connections a given site accepts. Using Oracle HTTP Server with `mod_oc4j`, you can limit the number of incoming requests by setting the `MaxClients` parameter in `httpd.conf`.

> **See Also:** "Configuring Oracle HTTP Server Directives" on page 5-11

Limiting Remote EJB Client Connections

To limit remote EJB client connections, you can use the global thread pool features that control the maximum number of threads that service incoming EJB clients. By configuring the `<global-thread-pool>` in `server.xml` to use two thread pools, you can set the parameter `cx-max` to limit remote EJB client connections.

> **See Also:** "Setting the OC4J Global Thread Pool Attributes in server.xml" on page 6-24

Limiting HTTP Connections with Standalone OC4J

If you are using standalone OC4J you can limit the number of active web users an OC4J site accepts concurrently by constraining the maximum allowable HTTP connections. Tuning parameters on a standalone OC4J can improve performance if there are a large number of concurrent users that the system cannot efficiently handle, or when there are limited resources which you cannot easily constrain.

To limit the HTTP connections, use the `max-http-connections` configuration element in `server.xml` and specify the attributes: `value`, `max-connections-queue-timeout`, and `socket-backlog`. The default `value` is 1000000, the default `max-connections-queue-timeout` is 10 seconds, and the default `socket-backlog` is 30.

For example, the following shows a line of `server.xml` that configures the maximum number of connections:

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100"/>
```

When you want messages to be redirected to a different URL when the maximum connections limit is reached, include the HTTP redirect URL.

For example, to redirect to `http://example.com/page.jsp`, add the following line to `server.xml`:

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100"> http://example.com/page.jsp
</max-http-connections>
```

> **See Also:** Appendix A, "Additional Information" in the *Oracle Application Server Containers for J2EE User's Guide* for information on `<max-http-connections>` attributes

Controlling Replication With Multiple OC4Js

This section covers the following:

- Controlling Web Application Replication
- Controlling Stateful Session EJB Replication

Controlling Web Application Replication

The replication features that provide for failover with Web sessions have a performance overhead. You should only use these features when their use is a requirement for the application or for the production environment.

You can disable replication for all applications running on OC4J using Application Server Control. From the OC4J Instance page select the Administration Link. Then, select the Replication Properties Link. On the Replication Properties page, deselecting the **Replicate session state** checkbox turns off Web replication for the OC4J Instance. This removes the `<cluster-config>` element from `global-web-application.xml` and disables OC4J Web replication for all applications running on the OC4J Instance.

If you do not want sessions to be replicated in a particular application, then remove the `<distributable/>` element from the application's `web.xml` file. This disables replication for the application even if OC4J has enabled replication.

With replication enabled, setting the `<distributable/>` element in `web.xml` can have significant performance overhead for applications that use sessions, since this configures the application to use session replication.

> **See Also:** *Oracle Application Server Containers for J2EE Servlet Developer's Guide*

Controlling Stateful Session EJB Replication

The replication features that provide for failover with stateful session EJBs have a performance overhead. Therefore, you should only use these features when their use is a requirement for your application or for your production environment.

> **See Also:** *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*

Performance Considerations for Deploying J2EE Applications

Many factors have an impact on the time it takes to deploy J2EE Applications on OC4J running in an Oracle Application Server environment.

This section covers the following:

- Deployment Performance During the Application Development Phase
- Deployment Performance During the Test and Production Phases

Deployment Performance During the Application Development Phase

The following development phase choices have an impact application deployment time for applications that are deployed to OC4J.

- **JVM flags** – The JVM `-server` flag is recommended for production use and is the default for OC4J when running in an Oracle Application Server environment. We have found `-server` usually improves performance in server environments. However, using the `-server` option increases the time required to restart a JVM and can require more memory.

  > **See Also:** "Setting Java Command Line Options (Using JVM and OC4J Performance Options)" on page 6-3

- **Heap requirements** – When you deploy large applications, if the deployment triggers JVM garbage collection, you may improve performance by increasing the size of the heap. Increasing the heap may provide enough memory so that the garbage collection is avoided. To increase the size of the heap, use either the `-mx` JVM option or set the `++AggressiveHeap` option. In addition, if the system is constrained for physical memory you may wish to shut down unused OC4J instances to reclaim physical memory.

  See Also:

  - "Setting the JVM Heap Size for OC4J Processes" on page 6-4
  - "Setting the JVM AggressiveHeap Option for OC4J Processes" on page 6-6

- **Application Type** – EJB applications require a compilation phase during deployment, and typically take longer to deploy than other type of J2EE applications.
- **Browser type** – The default configuration of Internet Explorer uses a buffer size of 8K. This size limitation can cause a delay in transmitting large files which can result in a significant performance degradation on deployments of large applications. We advise changing the configuration option to increase the buffer size. For detailed instructions on changing the buffer size, see the following site,

  http://support.microsoft.com/default.aspx?scid=kb;en-us;329781

  This issue is also present with Netscape 7.0 and Mozilla 1.0.2., however we are not aware of any workarounds. If you are using these browsers, you may decide to use a different browser or manually copy the `.ear` file to the local host and deploy using a command line tools (`admin.jar` or `dcmctl`). Netscape versions 4.79 and 7.1 do not exhibit this problem.
- **File system utilization** – Deploying applications involves file I/O to the local deployment directory. File I/O speeds may be impacted by the percentage utilization of the file system. Consult your platform documentation for recommendations about optimal utilization levels. However, many platform vendors recommend maintaining file system utilization below 90% for optimal performance.

Deployment Performance During the Test and Production Phases

The following test or production phase choices have an impact on the time it takes to deploy an application to OC4J.

- **Deployment Tool** – For production use with Oracle Application Server, you must deploy applications using either: `dcmctl`, Application Server Control, or JDeveloper. If you use `dcmctl` to deploy applications and need to perform multiple deploys or management commands, using the `dcmctl` shell mode provides a minor performance savings. Using the shell mode, the `dcmctl` client maintains a single client process for all the commands you run.
- **Repository Type** – If you are using an Oracle Application Server with a database repository, where the repository runs on a remote host, you may see slightly higher deployment times depending on the network latency at your site. Deploying to an Oracle Application Server with a local file-based repository usually is the most performant. Your choice of repository type should be driven by the availability and architectural requirements for your site and by application and deployment requirements.

  See Also: *Oracle Application Server 10g High Availability Guide*

- **Oracle HTTP Server Process State** – As a final phase of application deployment, if the Oracle HTTP Server is running, OPMN issues a command to restart the Oracle HTTP Server. This action updates the routing information for the newly deployed application. If you have a number of applications to deploy, and you are not running in a live production environment, you may wish to leave the Oracle HTTP Server down until after all applications are deployed. This avoids repeated restarts for the Oracle HTTP Server. However, restarting the Oracle HTTP Server only takes a few seconds, depending on the system speed, so the performance savings is not dramatic unless you are deploying a large number of applications.
- **OC4J Process State** – If the OC4J instance that you wish to deploy to is not started at the time the deploy command is issued, OPMN will start the instance and then shut it down when the deployment is complete. Again, these restart times are primarily significant when you are deploying multiple applications.
- **Heap requirements** – When you deploy large applications, the deployment may trigger JVM garbage collection. In memory-constrained environments, where you cannot increase the size of the heap to provide enough memory so that the garbage collection is avoided, then, typically, the only result of a garbage collection is an increase is the application deployment time (and an increase in response times for requests to the OC4J instance or request

timeouts). However, if the application being deployed is extremely large, the extended duration of the garbage collection may trigger OPMN to restart the OC4J instance.

To avoid OC4J restarts, increase the OPMN ping failure limit by setting values for the `no-reverseping-failed-ping-limit` and `reverseping-failed-ping-limit` parameters in `opmn.xml`. For example, set these values as follows:

```
<category id="restart-parameters">
   <data id="no-reverseping-failed-ping-limit" value="2"/>
   <data id="reverseping-failed-ping-limit" value="10"/>
</category>
```

The default value for `no-reverseping-failed-ping-limit` is 1 and the default value for `reverseping-failed-ping-limit` is 3.

7

Optimizing OracleAS Web Cache

This chapter provides guidelines for improving the performance of Oracle Application Server Web Cache (OracleAS Web Cache).

This chapter contains the following topics:

- Use Two CPUs for OracleAS Web Cache
- Configure Enough Memory for OracleAS Web Cache
- Make Sure You Have Sufficient Network Bandwidth
- Set a Reasonable Number of Network Connections
- Tune Network-Related Parameters
- Increase Cache Hit Rates
- Check Application Web Server and Web Cache Settings to Optimize Response Time

> **See Also:** *Oracle Application Server Web Cache Administrator's Guide* for more information about using OracleAS Web Cache.

Use Two CPUs for OracleAS Web Cache

OracleAS Web Cache can make best use of one or two CPUs. Because OracleAS Web Cache is an in-memory cache, it is rarely limited by CPU cycles. Additional CPUs do not increase performance significantly. However, the *speed* of the processors is critical—use the fastest CPUs you can afford.

Note that OracleAS Web Cache is limited by the available addressable memory. Additional memory can increase performance and scalability. See "Configure Enough Memory for OracleAS Web Cache" on page 7-3 for information about the amount of memory needed.

OracleAS Web Cache has two processes: one for the admin server and one for the cache server.

- The admin server process is used for configuring and monitoring OracleAS Web Cache. This process consumes very little CPU time. However, when viewing the statistics pages in OracleAS Web Cache Manager, the admin server process must query the cache server process to obtain the relevant metrics. Accessing the statistics pages frequently, or setting a high refresh rate on a statistics page can affect cache server performance.
- The cache server uses two threads: one to manage incoming connections and one to process requests. Because of this, two CPUs dedicated to OracleAS Web Cache are optimal.

For a cost-effective way to run OracleAS Web Cache, run it on a fast two-CPU computer with lots of memory. See the *Oracle Application Server Web Cache Administrator's Guide* for information about various deployment scenarios.

For a Web site with more than one OracleAS Web Cache instance, consider installing each instance on a separate two-CPU node, either as part of a cache cluster or as a standalone instance. When OracleAS Web Cache instances are on separate nodes, you are less likely to encounter operating system limitations, particularly in network throughput. For example, two caches on two separate two-CPU nodes are less likely to encounter operating system limitations than two caches on one four-CPU node.

Of course, if other resources are competing with OracleAS Web Cache for CPU usage, you should take the requirements of those resources into account when determining the number of CPUs needed. Although a separate node for OracleAS Web Cache is optimal, you can also derive a significant performance benefit from OracleAS Web Cache running on the same node as the rest of the application Web server.

Configure Enough Memory for OracleAS Web Cache

To avoid swapping documents in and out of the cache, it is crucial to configure enough memory for the cache. Generally, the amount of memory (maximum cache size) for OracleAS Web Cache should be set to at least 256 MB.

To be more precise in determining the maximum amount of memory required, you can take the following steps:

1. Determine what documents you want to cache, how many are smaller than 2 kilobytes (KB), and how many are larger than 2 KB. Determine the average size of the documents that are larger than 2 KB. Determine the expected peak load—the maximum number of documents to be processed concurrently.

   One way to do this is to look at existing Web server logs for one day to see what documents are popular. From the list of URLs in the log, decide which ones you want to cache. Retrieve the documents and get the size of each document.

2. Calculate the amount of memory needed. The way you calculate it may differ depending on the version of OracleAS Web Cache.

   The amount of memory that OracleAS Web Cache uses to store a document depends on the document size:

   - If a document is smaller than 2 KB, OracleAS Web Cache uses a buffer of 2 KB to store the HTTP body.
   - If a document is 2 KB or larger, OracleAS Web Cache uses buffers of 8 KB to store the HTTP body. For example, if a document is 42 KB, OracleAS Web Cache uses six 8 KB buffers to store the HTTP body.
   - Regardless of the size of the body, OracleAS Web Cache uses 8 KB to store the HTTP response header.

   Use the following formula to determine an estimate of the maximum memory needed:

   ```
   ( X * ( 2KB + 8KB ) ) + ( Y * (( [m/8] * 8KB ) + 8KB )) + basemem
   ```

   In the formula:

   - *X* is the number of documents smaller than 2 KB.
   - 2KB is the buffer size for the HTTP body for documents smaller than 2 KB.
   - 8KB is the buffer size for the HTTP response header.
   - Y is the number of documents that are 2 KB or larger.

- [*m*/8] is the ceiling of *m* (the average size, in kilobytes, of documents 2 KB or larger) divided by 8. A **ceiling** is the closest integer that is greater than or equal to the number.
- 8KB is the buffer size for the HTTP body for documents that are 2 KB or larger.
- 8KB is the buffer size for the HTTP response header.
- *basemem* is the base amount of memory needed by OracleAS Web Cache to process requests. This amount includes memory for internal functions such as lookup keys, connections to the application Web server to process cache misses, and timestamps. The amount needed depends on the number of *concurrent* requests and on whether or not the requests include Edge Side Includes (ESI). ESI is a markup language to enable partial-page caching of HTML fragments.

  For non-ESI requests, each concurrent request needs approximately 32 KB of memory. For example, to support 1000 concurrent requests, you need about 32 MB of memory.

  For ESI requests, each concurrent request needs roughly the following amount of memory:

  ```
  32KB + (number of ESI fragments * [8KB to 16KB])
  ```

  Because documents with more ESI fragments require more metadata for each fragment, use the higher number (16) for documents with 10 or more fragments. For example, for a document with 10 ESI fragments, use the following calculation:

  ```
  32KB + (10 * [16KB]) = 192KB
  ```

  That is, you need about 192 KB of memory for one 10-fragment document. To support 1000 concurrent requests, you need roughly 192 MB of memory.

For example, assume that you want to cache 5000 documents that are smaller than 2 KB and 2000 documents that are 2 KB or larger and that the larger documents have an average size of 54 KB. The documents do not use ESI. You expect to process 500 documents concurrently. Use the formula to compute the maximum memory:

```
(5000 * (2KB + 8KB) ) + ( 2000 * (( [54/8] * 8KB ) + 8KB )) + (500 * 32KB)
```

Using the formula, you need:

- 50,000 KB for the smaller documents.
- 128,000 KB for the larger documents. For the HTTP body, you need 56 KB (seven 8 KB buffers) for each document, given the average size of 54 KB. For the HTTP response header, you need 8 KB for each document.
- Approximately 16,000 KB for the base amount of memory needed to process 500 concurrent requests.

This results in an estimate of 194,000 KB of memory needed.

Note: Even though you specify that certain documents should be cached, not all of the documents are cached at the same time. Only those documents that have been requested and are valid are stored in the cache. As a result, only a certain percentage of the documents are stored in the cache at any given time. You may not need the maximum memory derived from the preceding formula.

---

3. Configure OracleAS Web Cache, specifying the result of the formula as the maximum cache size. Remember that the result is only an estimate.

   To specify the maximum cache size, take the following steps:

   a. In the navigator pane, select **Properties > Resource Limits.**

   b. On the Resource Limits page, select the cache and click **Edit.**

      The Edit Resource Limits dialog box appears.

   c. In the **Maximum Cache Size** field, enter the result of the formula.

   d. Click **Submit.**

   e. In the OracleAS Web Cache Manager main window, click **Apply Changes.**

4. Restart OracleAS Web Cache.
5. Using a simulated load or an actual load, monitor the cache to see how much memory it really uses in practice.

   The cache is empty when OracleAS Web Cache starts. For monitoring to be valid, make sure that the cache is fully populated. That is, make sure that the cache has received enough requests so that a representative number of documents are cached.

   The OracleAS Web Cache Statistics page (**Monitoring > Web Cache Statistics**) provides information about the current memory use and the maximum memory use. Note the following metrics in the Cache Overview table:

   - **Size of Documents in Cache** shows the current *logical* size of the cache, which is the size of the valid documents in the cache. For example, if the cache contains two documents, one 3 KB and one 50 KB, the **Size of Documents in Cache** is 53 KB, the total of the two sizes.
   - **Configured Maximum Cache Size** indicates the maximum cache size as specified in the Resource Limits page.
   - **Current Allocated Memory** displays the *physical* size of the cache, which is the amount of data memory allocated by OracleAS Web Cache for cache storage and operation. This number is always smaller than the process size shown by operating system statistics because the OracleAS Web Cache process, like any user process, consumes memory in other ways, such as instruction storage, stack data, thread, and library data.
   - **Current Action Limit** is 95% of the **Configured Maximum Cache Size.** This number is usually larger than the **Current Allocated Memory.**

   If the **Current Allocated Memory** is greater than the **Current Action Limit,** OracleAS Web Cache begins to use allocated but unused memory, and may begin garbage collection to free more memory. During garbage collection, OracleAS Web Cache removes the less popular and less valid documents from the cache in favor of the more popular and more valid documents to obtain space for new HTTP responses without exceeding the maximum cache size.

   If the **Current Allocated Memory** is close to or greater than the **Current Action Limit,** increase the maximum cache size to avoid swapping documents in and out of the cache. Use the Resource Limits page (**Properties > Resource Limits**) to increase the maximum cache size.

Make Sure You Have Sufficient Network Bandwidth

When you use OracleAS Web Cache, make sure that each node has sufficient network bandwidth to accommodate the throughput load. Otherwise, the network may be saturated but OracleAS Web Cache has additional capacity. For example, if an application generates more than 100 megabits of data per second, 10/100 Megabit Ethernet will likely be saturated.

If the network is saturated, consider using Gigabit Ethernet rather than 10/100 Megabit Ethernet. Gigabit Ethernet provides the most efficient deployment scenario to avoid network collisions, retransmissions, and bandwidth starvations. Additionally, consider using two separate network cards: one for incoming client requests and one for requests from the cache to the application Web server.

If system monitoring shows that the network is underutilized and throughput is less than expected, check whether or not the CPUs are saturated.

Set a Reasonable Number of Network Connections

It is important to specify a reasonable number for the maximum connection limit for the OracleAS Web Cache server. If you set a number that is too high, performance can be affected, resulting in slower response time. If you set a number that is too low, fewer requests will be satisfied. Strike a balance between response time and the number of requests processed concurrently.

To help determine a reasonable number, consider the following factors:

- The maximum number of clients that you intend to serve concurrently at any given time.
- The average size of a document and the average number of requests per document.
- Network bandwidth. The amount of data that can be transferred at any one time is limited by the network bandwidth. See "Make Sure You Have Sufficient Network Bandwidth" on page 7-7 for further information.
- The percentage of cache misses. Cache misses are forwarded to the application Web server. Those requests consume additional network bandwidth, resulting in longer response times, especially if a large percentage of requests are cache misses.
- How quickly a document is processed. Use a network monitoring utility, such as `ttcp`, to determine how quickly your system processes a document.

- The cache cluster member capacity, if you have a cache cluster environment. The capacity reflects the number of incoming connections from other cache cluster members. Set the cluster member capacity using the Clustering page **(Properties > Clustering)** of OracleAS Web Cache Manager.

Use various tools, such as those available with the operating system and with OracleAS Web Cache, to help determine the maximum number of connections. For example, the `netstat -a` command enables you to determine the number of established connections; the `ttcp` utility enables you to determine how fast a document is processed. The OracleAS Web Cache Manager provides statistics on hits and misses.

To set the maximum number of incoming connections, take the following steps:

1. In the navigator pane of OracleAS Web Cache Manager, select **Properties > Resource Limits.**
2. On the Resource Limits page, select the cache and click **Edit.**

   The Edit Resource Limits dialog box appears.
3. In the **Maximum Incoming Connections** field, enter the new value.
4. Click **Submit.**
5. In the OracleAS Web Cache Manager main window, click **Apply Changes.**

Do not set the value to an arbitrary high value. OracleAS Web Cache sets aside some resources for each connection, which could adversely affect performance. For many UNIX systems, 5000 connections is usually a reasonable number.

Connections on UNIX Platforms

On most UNIX platforms, each client connection requires a separate file descriptor. The OracleAS Web Cache server attempts to reserve the maximum number of file descriptors when it starts. If the `webcached` executable is run as `root`, you can increase this number. For example, for the Solaris Operating System you can increase the maximum number of file descriptors by setting the `rlim_fd_max` parameter. If `webcached` is not run as `root`, the OracleAS Web Cache server logs an error message and fails to start.

For information about running the `webcached` executable as the root user, see the *Oracle Application Server Web Cache Administrator's Guide*.

OracleAS Web Cache uses the following formula to calculate the maximum number of file descriptors to be used:

```
Max_File_Desc = Curr_Max_Conn + Total_WS_Capacity + Outgoing_Cluster_Conn + 100
```

In the formula:

- `Max_File_Desc` is the maximum number of file descriptors to be used.
- `Curr_Max_Conn` is the current maximum incoming connections limit for OracleAS Web Cache. Set the maximum number of incoming connections using the Resource Limits page (**Properties > Resource Limits**) of the OracleAS Web Cache Manager.

  In a cache cluster environment, `Curr_Max_Conn` also includes the cluster member capacity, which is the incoming connections from peer caches. Set the capacity using the Clustering page (**Properties > Clustering**) of the OracleAS Web Cache Manager.
- `Total_WS_Capacity` is the sum of the capacity for all configured application Web servers. Set the capacity using the Origin Servers page (**Origin Servers, Sites, and Load Balancing > Origin Servers**) of the OracleAS Web Cache Manager.

  In a cache cluster environment, the capacity is divided among the cache cluster members, using the following formula:

  ```
  Total_WS_Capacity = Sum_Web_Server_Capacity / n
  ```

  In the formula, `Sum_Web_Server_Capacity` is the sum of the capacity of all configured application Web servers; `n` is the number of cache cluster members. For example, assume you have two configured application Web Servers. `Web_Server_A` has a capacity of 200 and `Web_Server_B` has a capacity of 250. Also, assume you have a cluster with three caches. The `Total_WS_Capacity` is 150, as the following example calculates:

  ```
  Total_WS_Capacity = (200 + 250) / 3
  ```
- `Outgoing_Cluster_Conn` is the total number of outgoing connections to peer caches in a cache cluster. This value is zero if you do not have a cache cluster. To compute this value, use the following formula:

  ```
  Outgoing_Cluster_Conn = Sum_Cluster_Capacity / (n-1)
  ```

  In the formula, `Sum_Cluster_Capacity` is the sum of the capacity of all other OracleAS Web Caches in a cluster; `n` is the number of cache cluster members.

For example, assume you have a cluster with three caches. `Cache_A` has a capacity of 100, `Cache_B` has a capacity of 150, and `Cache_C` has a capacity of 200. The `Outgoing_Cluster_Conn` for `Cache_A` is 175, computed as follows:

```
Outgoing_Cluster_Conn = (150 + 200) / (3-1)
```

To set the capacity of caches in a cluster, select **Properties > Clustering** from the navigator pane of OracleAS Web Cache Manager.

- `100` is the number of connections reserved for internal use by OracleAS Web Cache.

Connections on Windows

On Windows, the number of file handles as well as socket handles is limited only by available kernel resources, more precisely, by the size of paged and non-paged pools. However, the number of active TCP/IP connections is restricted by the number of TCP ports the system can open.

The default maximum number of TCP ports is set to 5000 by the operating system. Of those, 1024 are reserved by the kernel. To modify the maximum number of ports, edit the Windows registry. The Windows operating systems allow up to 65536 ports.

To change the default, you must add a new value to the following registry key:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

Add a new value, specifying the following:

- Value Name: MaxUserPort
- Data Type: REG_DWORD
- Data: An integer less than 65536 - 1024

The total of the maximum number of incoming connections and cluster member capacity should not be set to a number greater than the number of TCP ports minus 1024. Set the maximum number of incoming connections using the Resource Limits page (**Properties > Resource Limits**) of the OracleAS Web Cache Manager. Set the cluster member capacity using the Clustering page (**Properties > Clustering**).

On Windows platforms, OracleAS Web Cache does not attempt to reserve file handles or to check that the number of current maximum incoming connections is less than the number of TCP ports.

Tune Network-Related Parameters

Besides the number of network connections, other network-related parameters for OracleAS Web Cache, the application Web server, and the operating system can affect response time. In most situations, the default settings are sufficient.

If response time is slow, you should tune OracleAS Web Cache, the application Web server, and operating system parameters that affect connections, as explained in this section.

For OracleAS Web Cache, check the values of the following settings:

- **Keep-Alive Timeout:** The amount of time a network connection is left open after OracleAS Web Cache sends a response to a browser. Keep-Alive allows an HTTP client to send multiple requests to OracleAS Web Cache using the same network connection. By default, the connection is left open for five seconds, which is typically enough time for the browser to send subsequent requests to OracleAS Web Cache using the same connection.

  If the network between the browser and OracleAS Web Cache is slow, consider increasing the timeout, perhaps up to 30 seconds.

  If you receive the following error, either increase the maximum incoming connections for OracleAS Web Cache or lower the **Keep-Alive Timeout:**

  ```
  11313: The cache server reached the maximum number of allowed incoming
  connections. Listening is temporarily suspended.
  ```

  With a heavy load, such as during stress-testing, if clients continuously send one request and then disconnect, set the **Keep-Alive Timeout** to 0. With this value, OracleAS Web Cache closes the connection as soon as the request is completed, to free up resources.

  Set the **Keep-Alive Timeout** value in the Network Timeouts page (**Properties > Network Timeouts**).

- **Origin Server Timeout:** The amount of time for the application Web server to generate a response to OracleAS Web Cache. If the application Web server or proxy server is unable to generate a response within that time, OracleAS Web Cache sends a network apology page to the browser.

  Usually, this value should be equal to the response time of the slowest document served by the application Web Server. If the value is too low, long-running requests will timeout before the response is complete. If the value is too high and the application Web server hangs for some reason, it will take longer for OracleAS Web Cache to failover to another application Web server.

Set this value in the Network Timeouts page (**Properties > Network Timeouts**).

For the application Web server, check the values of the following settings in the application Web server's configuration file (`httpd.conf`). (These particular parameter names are specific to the Oracle HTTP Server.)

- `KeepAlive`: Whether to allow persistent connections. Persistent connections allow a client to send multiple sequential requests through the same connection.

  Make sure `KeepAlive` is enabled. This can improve performance because the connection is set up only once and is kept open for subsequent requests from the same client.

- `KeepAliveTimeout`: The time a connection is left open to wait for the next request from the same client. If requests are primarily from OracleAS Web Cache, you can set this value fairly high. A reasonable value is 30 seconds.
- `MaxKeepAliveRequests`: The maximum number of requests to allow during a persistent connection. Set to 0 to allow an unlimited number of requests.
- `MaxClients`: The maximum number of clients that can connect to the application Web server simultaneously.

  If `KeepAlive` is enabled for the application Web server, you may require more concurrent httpd server processes, and you may need to set the `MaxClients` directive to a higher value.

  If client requests have a short response time, you may be able to improve performance by setting `MaxClients` to a lower value. However, when this value is reached, no additional processes will be created, causing other requests to fail.

  The `MaxClients` limit on the application Web server should be greater than or equal to the application Web server capacity as set through the OracleAS Web Cache Manager.

For the operating system, check the TCP time-wait setting. This setting controls the amount of time that the operating system holds a port, not allowing new connections to use the same port.

On the Solaris Operating System, for example, check the `tcp_time_wait_interval` setting, using the following command:

```
ndd -get /dev/tcp tcp_time_wait_interval
```

On Windows 2000, for example, check the value of the `TcpTimeWaitDelay` parameter in the following key in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

This setting is usually only an issue during stress testing, if you continuously open more TCP/IP connections from one client computer. In this situation, lower the TCP time-wait setting. In real world deployments, this is rarely an issue because it is unlikely that a single client will generate a huge number of connections.

See Also: "Configuring Oracle HTTP Server Directives" on page 5-11

Increase Cache Hit Rates

A **cache hit** is a Web browser request that can be satisfied from documents stored in the cache. A **cache miss** is a Web browser request that cannot be satisfied from documents stored in the cache and must be forwarded to the application Web server.

If the ratio of cache hits to cache misses is low, consider the following ways to raise the cache hit rate:

- Use cookies and URL parameters to increase cache hit rates.

  OracleAS Web Cache can cache different versions of a document with the same URL, based on request cookies or headers. To use this feature, applications may need to implement some simple change, such as creating a cookie or header that differentiates the documents.

  On the opposite end of the spectrum, some applications contain some insignificant URL parameters, which can lead to different URLs representing essentially the same content. If the documents are cached under their full URLs, the cache hit/miss ratio becomes very low. You can configure OracleAS Web Cache to ignore the non-differentiating URL parameter values, so that a single document is cached for different URLs, greatly increasing cache hit rates.

  Sometimes the content for a set of documents is nearly identical, but not exactly identical. For example, the documents may contain hyperlinks composed of the same URL parameters with different session-specific values, or they may include some personalized strings in the document text, such as welcome greetings or shopping cart totals. You can configure OracleAS Web Cache to store a single copy of the document with placeholders for the embedded URL parameters or the personalized strings, and to dynamically substitute the correct values for the placeholders when serving the document to clients.

For more information on multiple version documents, sessions, ignoring URL parameter values, and simple personalization, see Chapter 2, "Caching Concepts," of the *Oracle Application Server Web Cache Administrator's Guide*.

- Use redirection to cache entry documents.

  For some popular site entry documents, such as "/", that typically require session establishment, session establishment effectively makes the document non-cacheable to all new users without a session. To cache these documents while preserving session establishment, you can either:

  – Create a blank document that provides session establishment for all initial requests and redirects to the actual popular document. Subsequent redirected requests to the popular document will specify the session, enabling the popular document to be served from the cache.

  – Use a JavaScript that sets a session cookie for the popular documents.

  For more information on configuring caching rules for documents requiring session establishment, see Chapter 9, "Creating Caching Rules," of the *Oracle Application Server Web Cache Administrator's Guide*.

- Use partial page caching where possible.

  Many Web documents, such as pages generated by OracleAS Portal, are composed of fragments with unique caching properties. For these pages, full-page caching is not feasible. However, OracleAS Web Cache provides partial page caching using Edge Side Includes (ESI). With ESI, you can divide each Web page into a template and multiple fragments that can, in turn, be further divided into templates and lower level fragments. Each fragment or template is stored and managed independently; a full page is assembled from the underlying fragments upon request. Fragments can be shared among different templates, so that common fragments are not duplicated to waste cache space. Sharing can also greatly reduce the number of updates required when fragments expire.

  Depending on the application, updating a fragment can be cheaper than updating a full page. In addition, each template or fragment can have its own unique caching policies such as expiration, validation, and invalidation, so that each fragment in a full Web page can be cached as long as possible, even when some fragments are not cached or are cached for a much shorter period of time.

- Use ESI variables for improved cache hit / miss ratio for personalized pages.

  Personalized information often appears in Web pages, making them unique for each user. For example, many Web pages contain tens or hundreds of hyperlinks embedding application session IDs. To resolve this, create your ESI pages with variables. Because variables can resolve to different pieces of request information or response information, the uniqueness of templates and fragments can be significantly reduced. This, in turn, results in better cache hit / miss ratios.

Check Application Web Server and Web Cache Settings to Optimize Response Time

If you have not configured the application Web server or the cache correctly, response time may be slower than anticipated. This section summarizes much of the information presented in this chapter.

If the application Web server is responding more slowly than expected or if the application Web server is not responding to requests from the cache because it has reached its capacity, check the application Web server and OracleAS Web Cache settings.

First, check the following:

- Caching rules: Make sure that you are caching the appropriate objects. Are there popular objects that you should cache but are not caching? Use the Popular Requests page (**Monitoring > Popular Requests**) to see a list of the most popular requests and to check that those objects are being cached. Also, see "Increase Cache Hit Rates" on page 7-13 for information on increasing the ratio of cache hits to cache misses.
- Priority rankings of the caching rules: Give frequently accessed non-cacheable documents a higher priority than cacheable documents. Give frequently accessed cacheable documents the lowest priority. Note that parsing of caching rules may be expensive if a large number of rules are defined.
- Compression: If the network is a bottleneck for the client, compressing documents as they are cached will relieve some of the congestion on the network because compressed documents are smaller.

Then, check the following:

- The application Web server configuration, particularly the `MaxClients`, `KeepAlive`, `KeepAliveTimeout`, and `MaxKeepAliveRequests` settings.

The `MaxClients` limit on the application Web server should be greater than or equal to the application Web server capacity as set through the OracleAS Web Cache Manager.

See "Tune Network-Related Parameters" on page 7-11 for more information.

- The application Web server capacity as set using the Origin Servers page (**Origin Servers, Sites, and Load Balancing > Origin Servers**) of the OracleAS Web Cache Manager. See the *Oracle Application Server Web Cache Administrator's Guide* for information about setting application Web server capacity.

Then, if the application Web server is still busier than anticipated, it may mean that the cache cannot process the requests and is routing more requests to the application Web server. Check the following OracleAS Web Cache settings in the OracleAS Web Cache Manager:

- The number of cache connections. Check **Maximum Incoming Connections** in the Resource Limits page (**Properties > Resource Limits**). See "Set a Reasonable Number of Network Connections" on page 7-7 for more information.
- The memory size for the cache. Check **Maximum Cache Size** in the Resource Limits page (**Properties > Resource Limits**). See Configure Enough Memory for OracleAS Web Cache on page 7-3 for more information.
- The cache cluster capacity. In a cache cluster, if cluster capacity is too low, a cache may not receive a response for owned content from a peer cache in the specified interval. As a result, the request is sent to the application Web server. Check **Capacity** in the Clustering page (**Properties > Clustering**). See the *Oracle Application Server Web Cache Administrator's Guide* for more information.

If the settings for the application Web server and OracleAS Web Cache are set correctly, but the response times are still higher than expected, check system resources, especially:

- Network bandwidth. See "Make Sure You Have Sufficient Network Bandwidth" on page 7-7 for more information.
- CPU usage. See "Use Two CPUs for OracleAS Web Cache" on page 7-2 for more information.

8

Optimizing PL/SQL Performance

This chapter discusses the techniques for improving PL/SQL performance in Oracle Application Server.

This chapter contains the following sections:

- PL/SQL Performance in Oracle Application Server - Overview
- Process-Based and Thread-Based Operation in Oracle HTTP Server
- Performance Tuning Issues for mod_plsql
- Performance Tuning Areas in mod_plsql
- Using Caching with PL/SQL Based Web Applications
- Tuning File System Cache to Improve Caching Performance
- Oracle HTTP Server Directives

PL/SQL Performance in Oracle Application Server - Overview

This chapter describes several techniques to improve the performance of PL/SQL based Web applications in Oracle Application Server (Oracle Application Server).

Table 8–1 lists recommendations for Database Access Descriptor (DAD) parameters and settings. By default, these DAD parameters are specified in the file `dads.conf`. On UNIX systems, this is in the `$ORACLE_HOME/Apache/modplsql/conf` directory. On Windows systems, by default, this file is in the directory `%ORACLE_HOME\Apache\Apache\modplsql\conf` directory. The file `dads.README` in this directory describes the DAD parameters in detail.

Table 8–1 Database Access Descriptor (DAD) Parameters Recommended Setting Summary

Parameter	Recommended Setting
`PlsqlAlwaysDescribeProcedure`	Set this to `off` for best performance. Default Value: `off`
`PlsqlDatabaseConnectString`	For newer DADs, use the `ServiceNameFormat`. Use the `SIDFormat` only for backward compatibility. Note: for HA configurations of the database, it is recommended that the connect string parameter gets resolved through an LDAP lookup.
`PlsqlFetchBufferSize`	For multibyte character sets like Japanese or Chinese, setting this to 256 should provide better performance Default Value: 128
`PlsqlIdleSessionCleanupInterval`	Increasing this parameter allows pooled database connections to remain available, in the pool, for the specified time Default Value: 15 (minutes) See Also: "What Happens to the mod_plsql Connection Pool when the Database Restarts?" on page 8-8
`PlsqlLogEnable`	This parameter should be set to `Off` unless recommended by Oracle support for debugging purposes Default Value: `off`

Table 8–1 (Cont.) Database Access Descriptor (DAD) Parameters Recommended Setting Summary

Parameter	Recommended Setting
`PlsqlMaxRequestsPerSession`	If the PL/SQL based Web application does not leak resources or memory, this parameter can be set to a higher value (for example, 5000). Default Value: 1000 See Also: "Closing Pooled Database Sessions" on page 8-7 and "Connection Pooling Tips and Oracle HTTP Server Configuration" on page 8-10
`PlsqlNLSLanguage`	Set this parameter to match the database Globalization Support parameters to eliminate overheads in character set conversions in Oracle Net Services
`PlsqlSessionStateManagement`	Set this parameter to `StatelessWithFastResetPackageState` if the database is 8.1.7.2 or above.

Table 8–2 lists `mod_plsql` caching options and the sections that describe these caching options.

Table 8–2 Caching Options

Option	Description
Expires Technique	Best performance - for content that changes predictably See Also: "Using the Expires Technique" on page 8-24
Validation technique	Good performance - for content that changes unpredictably See Also: "Using the Validation Technique" on page 8-19
System-level caching	Improves performance by caching one copy for everyone on system See Also: "System- and User-level Caching with PL/SQL Based Web Applications" on page 8-26

See Also:

- Appendix A, "Performance Metrics" for information on `mod_plsql` metrics
- Chapter 6" Oracle HTTP Server Modules" in the *Oracle HTTP Server Administrator's Guide* for details on the DAD Parameters shown in Table 8–1
- *Oracle Application Server 10g mod_plsql User's Guide*
- *Oracle Application Server 10g PL/SQL Web Toolkit Reference*

Process-Based and Thread-Based Operation in Oracle HTTP Server

This chapter describes PL/SQL performance issues that apply on platforms where the Oracle HTTP Server is process-based and thread-based. On a process-based Oracle HTTP Server, such as those running on UNIX-based platforms, each process servers all types of HTTP requests, including servlets and PL/SQL, static files. On a thread-based Oracle HTTP Server, such as Windows-based platforms, there is just one Oracle HTTP Server process with multiple threads within the process; individual threads can be used serve all types of HTTP requests.

Note: In some cases in this chapter we make references to performance optimizations that apply for PL/SQL based Web applications where the distinction between platforms, either process-based or thread based is significant.

Performance Tuning Issues for mod_plsql

When tuning `mod_plsql` to improve the performance of PL/SQL based Web applications, it is important to be familiar with some `mod_plsql` internals. This section presents a basic overview of some `mod_plsql` functionality.

This section covers the following topics:

- Connection Pooling with mod_plsql
- Closing Pooled Database Sessions
- What Happens to the mod_plsql Connection Pool when the Database Restarts?

Connection Pooling with mod_plsql

The Database Server connection pooling logic supplied with `mod_plsql` can be best explained with an example.

For example, consider the following typical scenario:

1. The Oracle HTTP Server listener is started. There are no database connections in the connection pool maintained by `mod_plsql`.
2. A browser makes a `mod_plsql` request (R1) for Database Access Descriptor (DAD) D1.
3. One of the Oracle HTTP Server processes (`httpd` process P1) starts servicing the request R1.
4. `mod_plsql` in process P1 checks its connection pool and finds that there are no database connections in its pool for that user request.
5. Based on the information in DAD D1, `mod_plsql` in process P1 opens a new database connection, services the PL/SQL request, and adds the database connection to its pool.
6. From this point on, all subsequent requests to process P1 for DAD D1 can now make use of the database connection pooled by `mod_plsql`.
7. If a request for DAD D1 gets picked up by another process (process P2), then `mod_plsql` in process P2 opens its own database connection, services the request, and adds the database connection to its pool.
8. From this point on, all subsequent requests to process P2 for DAD D1 can now make use of the database connection pooled by `mod_plsql`.
9. Now, assume that a request R2 is made for DAD D2 and this request gets routed to process P1.
10. `mod_plsql` in process P1 does not have any database connections pooled for DAD D2, and a new database session is created for DAD D2 and pooled after servicing the request. Process P1 now has two database connections pooled, one for DAD D1 and another for DAD D2.

    The important details in the example shown in steps 1-10 are:

    a. Each Oracle HTTP Server process serves all types of requests, such as static files requests, servlet requests, and `mod_plsql` requests. There is no control on which Oracle HTTP Server process services the next request.

b. One Oracle HTTP Server process cannot use or share the connection pool created by another process.

Note: On Windows systems, the Oracle HTTP Server is just one process. Therefore, the connection pool is shared and can be used across threads.

---

c. Each Oracle HTTP Server process pools at most one database connection for each DAD.

d. User sessions are switched within a pooled database connection for a DAD. For DADs based on Oracle Application Server Single Sign-On (SSO), proxy authentication is used to switch the user session. For non-SSO users, using HTTP basic authentication with the username and password not in the DAD, users are re-authenticated on the same connection.

e. Multiple DADs may point to the same database instance, but database connections are not shared across DADs even within the same process.

f. Unused DADs do not result in any database connections.

In the worst case scenario, the total number of database connections pooled by `mod_plsql` is a factor of the total number of active DADs multiplied by the number of Oracle HTTP Server (`httpd`) processes running at any given time for a single Oracle Application Server instance. If you have configured the Oracle HTTP Server processes to a high number, you need to configure the backend database to handle a corresponding number of database sessions, and remember that this configuration value needs to be multiplied times the number of Oracle Application Server Instances that use the backend database.

For example, if there are three Oracle Application Server instances configured to spawn a maximum of 50 `httpd` processes each, plus two active DADs, you need to set up the database to allow 300 (`3*50*2`) sessions. This number does not include any sessions that are needed to allow other Web applications to connect.

On UNIX systems, database connections cannot be shared across `httpd` processes, and process-based platforms have more of a *Connection Reuse* feature than *Connection Pooling*. Note that this is an artifact of the process-model in Oracle HTTP Server.

Note: Refer to "Two-Listener Strategy" on page 8-14 if the number of database sessions is a concern for details on how to address this problem.

On Windows systems, the Oracle HTTP Server runs as a single process, and the connection pool is shared. Therefore, the "Two-Listener Strategy" on page 8-14 does not apply to Windows systems.

Closing Pooled Database Sessions

Pooled database sessions are closed under the following circumstances:

1. When a pooled connection has been used to serve a configured number of requests.

   By default each connection pooled by `mod_plsql` is used to service a maximum of 1000 requests and then the database connection is shut down and reestablished. This is done to make sure that any resource leaks in the PL/SQL based Web application, or in the Oracle client server side, do not adversely affect the system. To change the default value of 1000 by tuning the DAD configuration parameter `PlsqlMaxRequestsPerSession`.

2. When a pooled connection has been idle for an extended period of time.

   By default, each pooled connection gets automatically cleaned up after 15 minutes of idle time. This operation is performed by the cleanup thread in `mod_plsql`. For heavily loaded sites, each connection could be used at least once every 15 minutes and the connection cleanup might not happen for a long period of time. In such a case, the connection would be cleaned up based on the configuration value of `PlsqlMaxRequestsPerSession`. Change the default value of 15 minutes by tuning the `mod_plsql` configuration parameter `PlsqlIdleSessionCleanupInterval`. Consider increasing the default for better performance in cases where the site is not heavily loaded.

3. On UNIX systems, when the Oracle HTTP Server process goes down.

   On UNIX systems, the Oracle HTTP Server configuration parameter `MaxRequestsPerChild` governs when an Oracle HTTP Server process will be shut down. For example, if this parameter is set to 5000, each Oracle HTTP Server process would serve exactly 5000 requests before it is shut down. Oracle HTTP Server processes could also start up and shut down as part of Oracle HTTP Server maintenance based on the configuration parameters

MinSpareServers, MaxSpareServers, and MaxClients. For mod_plsql connection pooling to be effective, it is extremely important that Oracle HTTP Server in Oracle Application Server be configured such that each Oracle HTTP Server process remains active for some period of time. An incorrect configuration of Oracle HTTP Server could result in a setup where Oracle HTTP Server processes are heavily started up and shut down. Such a configuration would require that each new Oracle HTTP Server process replenish the connection pool before subsequent requests gain any benefit of pooling.

See Also: Chapter 6 "Oracle HTTP Server Modules" in the *Oracle HTTP Server Administrator's Guide*

What Happens to the mod_plsql Connection Pool when the Database Restarts?

This depends primarily on the amount of time the database is shut down. If the database is restarted after more than 15 minutes from being shut down, the users do not experience any problems when trying to use the Oracle Application Server listener. This is because the cleanup thread in mod_plsql cleans up database sessions that are unused for more than 15 minutes. The time specified for cleaning up idle sessions is tunable using the PlsqlIdleSessionCleanupInterval, configuration parameter (the default value is 15 minutes).

If the database is restarted in less than 15 minutes, then a few initial requests return with errors, but the system quickly becomes usable again. The number of requests that experience failure is equal to the number of connections that were pooled by mod_plsql.

See Also: Table 8–1, " Database Access Descriptor (DAD) Parameters Recommended Setting Summary"

Performance Tuning Areas in mod_plsql

While using mod_plsql, there are three areas that affect performance and scalability:

- PL/SQL Based Web Application Development Considerations and Programming Tips
- Connection Pooling Tips and Oracle HTTP Server Configuration
- Tuning the Number of Database Sessions

PL/SQL Based Web Application Development Considerations and Programming Tips

PL/SQL Gateway users should consider the following topics when developing PL/SQL based Web applications:

1. **Manage the use of Database Access Descriptors (DADs)**

   Try to restrict the number of DADs that each Oracle Application Server node uses.

   > **Note:** Performance is not affected if there are DADs that are not being used.

2. **Use of Nested Tables**

   PL/SQL provides the ability to create tables. To build PL/SQL tables, you build a table that gives the datatype of the table, as well as the index of the table. The index of the table is the binary integer ranging from -2147483647 to +2147483647. This table index option is known as *sparsity*, and allows meaningful index numbers such as customer numbers, employee number, or other useful index keys. Use PL/SQL tables to process large amounts of data.

   PL/SQL provides TABLE and VARRAY (variable size array) collection types. The TABLE collection type is called a nested table. Nested tables are unlimited in size and can be sparse, which means that elements within the nested table can be deleted using the DELETE procedure. Variable size arrays have a maximum size and maintain their order and subscript when stored in the database. Nested table data is stored in a system table that is associated with the nested table. Variable size arrays are suited for batch operations in which the application processes the data in batch array style. Nested tables make for efficient queries by storing the nested table in a storage table, where each element maps to a row in the storage table.

3. **Use procedure naming overloading with caution**

   PL/SQL based Web applications should use the procedure name overloading feature with caution. It is best if procedure name overloading is avoided by having multiple procedures with different names.

4. **Consider rewriting applications where there is significant overhead in determining the type parameters**

   PL/SQL based Web applications should be aware of the overhead in trying to execute procedures where the URL does not provide enough details to know about the type of the parameter, such as scalar or array. In such cases, the first attempt to execute a procedure fails and the procedure signature needs to be described before it can be executed by `mod_plsql`.

5. **Use procedures with 2-parameter style flexible parameter passing**

   Procedures should make use of the more performant 2-parameter style flexible parameter passing rather than the 4-parameter style parameter passing

   See Also: *Oracle Application Server 10g PL/SQL Web Toolkit Reference*

Connection Pooling Tips and Oracle HTTP Server Configuration

Consider the following topics when configuring connection pooling with Oracle HTTP Server:

1. Using the default connections pooling and setting values for `PlsqlMaxRequestsPerSession`

   Creating new database connections is an expensive operation and it is best if every request does not have to open and close it own database connections. The optimal technique is to make sure that database connections opened in one request are reused in subsequent requests. In some rare situations, where a database is accessed very infrequently and performance is not a major concern, connection pooling can be disabled. For example, if the administrator accesses a site infrequently to perform some administration tasks, then the DAD used to access the administrator applications can choose to disable connection pooling. To disable connection pooling, set the DAD parameter `PlsqlMaxRequestsPerSession` to the value 1.

   Note: Setting `PlsqlMaxRequestsPerSession` to the value 1 reduces the number of available database sessions and may impact performance.

2. On UNIX systems, Oracle HTTP Server configuration should be properly tuned so that once processes are started up, the processes remain up for a while. Otherwise, the connection pooling in `mod_plsql` is rendered useless. The Oracle Application Server listener should not have to continually start up and shut down processes. A proper load analysis should be performed of the site to determine what the average load on the Web site. The Oracle HTTP Server configuration should be tuned such that the number of `httpd` processes can

handle the average load on the system. In addition, the configuration parameter `MaxClients` in the `httpd.conf` file should be able to handle random load spikes as well.

3. On UNIX systems, Oracle HTTP Server processes should be configured so that processes are eventually killed and restarted. This is required to manage any possible memory leaks in various components accessed through the Oracle HTTP Server. This is specifically required in `mod_plsql` to ensure that any database session resource leaks do not cause a problem. Make sure that `MaxRequestsPerChild` configuration parameter is set to a high number. For PL/SQL based Web applications, this should not be set to `0`.

> **Note:** On Windows systems, to assure that database session resource leaks do not cause problems, the Oracle HTTP Server needs to be periodically restarted.

4. For heavily loaded sites, the Oracle HTTP Server configuration parameter `KeepAlive` should be disabled. This ensures that each process is available to service requests from other clients as soon as a process is done with servicing the current request. For sites which are not heavily loaded, and where it is guaranteed that the number of Oracle HTTP Server processes are always greater than the number of simultaneous requests to the Oracle Application Server listener, enabling the `KeepAlive` parameter results in performance improvements. In such cases, make sure to tune the `KeepAliveTimeout` parameter appropriately.

5. You may want to lower the value of `Timeout` in the Oracle HTTP Server configuration. This ensures that Oracle HTTP Server processes are freed up earlier if a client is not responding in a timely manner. Do not set this value too low, otherwise slower responding clients could time out.

6. Most Web sites have many static image files which are displayed in each screen for a consistent user interface. Such files rarely change and you can reduce a considerable load on the system by tagging each image served by the Oracle Application Server listener with `mod_expires`. You should also consider front-ending your Web site with Oracle Application Server Web Cache.

   - How do I know if the Web site can benefit from the use of `mod_expires`?

     – Use Netscape, or any browser that allows you to view page caching information, and visit several heavily-accessed Web pages on the site. On each page, right click the mouse and select `View Info` from the pop up menu (or the equivalent command for your browser). If the top panel in the page information window lists many different images and static content, then the site could benefit from the use of `mod_expires`.

– You can also check the Oracle HTTP Server access logs to see what percentage of requests result in HTTP 304 (Not Modified) status. Use the `grep` utility to search for `304` in the `access_log` and divide this resulting number of lines by the total number of lines in the `access_log`. If this percentage is high, then the site could benefit from the use of `mod_expires`.

- How do I tag static files with the expires header?

– Locate the `Location` directive used to serve your static image files. Add the `ExpiresActive` and `ExpiresDefault` directives to it.

```
Alias /images/ "/u01/app/oracle/myimages/"
<Directory "/u01/app/oracle/myimages/">
     AllowOverride None
     Order allow, deny
     Allow from all
     ExpiresActive On
     ExpiresDefault A2592000
</Directory>
```

The browser caches all static files served off the `/images` path for 30 days from now. Refer to *Oracle HTTP Server Administrator's Guide* for more details.

– If you are using Oracle Application Server Web Cache, these files can be cached in memory with the use of the `Surrogate-Control` header. For example:

```
Alias /images/ "/u01/app/oracle/myimages/"
<Directory "/u01/app/oracle/myimages/">
     AllowOverride None
     Order allow, deny
     Allow from all
     ExpiresActive On
     ExpiresDefault A2592000
     <Files *>
        Header set Surrogate-Control 'max-age=2592000'
     </Files>
</Directory>
```

Refer to the *Oracle Application Server Web Cache Administrator's Guide* for more details on the `Surrogate-Control` header.

- How do I know if the static files are being tagged with the `Expires` header?
  - Using Netscape, or the browser of your choice, clean up all the cached files in the browser.
  - Visit a Web page that should have images tagged with the `Expires` header. Right click the mouse on the page and select View Info, from the pop up menu. or use the equivalent command for your browser.
  - In the top panel of the page information, select an image that should be tagged with the `Expires` header.
  - Review the information displayed in the bottom panel. The `Expires` header should be set to a valid date. If this entry is `No date given`, then the file is not being tagged with the `Expires` header.

See Also: *Oracle Application Server 10g mod_plsql User's Guide* for details on using `mod_expires`.

Tuning the Number of Database Sessions

Consider the following topics when tuning the number of database sessions:

1. The `processes` and `sessions` parameters in the Oracle `init$SID.ora` configuration file should be set so that Oracle is able to handle the maximum number of database sessions. This number should be proportional to the number of DADs times the maximum number of Oracle HTTP Server processes, times the number of Oracle Application Server Instances.
2. Using a two-listener strategy or using a shared server reduces the number of database sessions. See "Two-Listener Strategy" on page 8-14.
3. On UNIX platforms, the connection pool is not shared across Oracle HTTP Server processes. For this reason, it is recommended that the application use as few DADs as possible.
4. Front ending your Oracle HTTP Server with Oracle Application Server Web Cache reduces the requirement to have a high number of processes for your HTTP configuration, resulting in lesser number of database sessions.

Note: This is only beneficial when Oracle HTTP Server is front-ended with Oracle Application Server Web Cache and Oracle Application Server Web Cache caches static content. To test that Oracle Application Server Web Cache is caching static content, see item 6 in the section, "Connection Pooling Tips and Oracle HTTP Server Configuration" on page 8-10.

Two-Listener Strategy

On platforms where the Oracle HTTP Server is process-based, such as all UNIX-based platforms, each process serves all types of HTTP requests, including servlets, PLSQL, static files, and CGI. In a single Oracle Application Server listener setup, each `httpd` process maintains its own connection pool to the database. The maximum number of database sessions is governed by the setting in `httpd.conf` configuration file for `StartServers`, `MinSpareServers`, and `MaxSpareServers`, plus the load on the system. This architecture does not allow for tuning the number of database sessions based on the number of `mod_plsql` requests. To tune the number of database sessions based on the number of `mod_plsql` requests, install a separate HTTP listener for `mod_plsql` requests only. This approach greatly reduces the number of database sessions that are needed to serve `mod_plsql` requests.

For example, assume a main Oracle Application Server listener is running on port 7777 of mylsnr1.mycompany.com. First, you can install another Oracle Application Server listener on port 8888 on mylsnr2.mycompany.com. Next, redirect all `mod_plsql` requests made to mylsnr1.mycompany.com:7777 to the second listener on mylsnr2.mycompany.com:8888. Review the following steps:

1. To redirect all PL/SQL requests for `mylsnr1.mycompany.com:7777` to `mylsnr2.mycompany.com:8888`, make the following configuration changes:

   a. For the Oracle Application Server listener running on Port 7777, edit `ORACLE_HOME/Apache/modplsql/conf/plsql.conf` file. Comment out the following line by putting a # in front of the line:

   ```
   #LoadModule plsql_module...
   ```

   b. Copy the DAD location used to service PL/SQL requests in `mylsnr1.mycompany.com` to the configuration file `$ORACLE_HOME/Apache/modplsql/conf/dads.conf` in `mylsnr2.mycompany.com`.

   Comment out the DAD location configuration parameters on `mylsnr1.mycompany.com` by prepending the line with a "#" character.

```
#<Location /pls/portal>
#...
#</Location>
```

c. Configure this listener to forward all `mod_plsql` requests for this DAD location to the second listener by adding the following line in `dads.conf`:

```
ProxyPass /pls/portal http://mylsnr2.mycompany.com:8888/pls/portal
```

Repeat the configuration procedures for all DAD Locations.

2. Because the PL/SQL procedures generate URLs that are displayed in the browser, it is important that all URLs are constructed without any references to the internal `mod_plsql` listener on mylsnr2.mycompany.com:8888. Depending on how the URLs are being generated in the PL/SQL based Web application, there are three options:

   - If the URLs are hard-coded into the application, make sure that they are always generated using the hard-coded values as HOST=mylsnr1.mycompany.com and PORT=7777. No change would be required for this scenario.
   - If the PL/SQL based Web applications always use the CGI environment variables `SERVER_NAME` and `SERVER_PORT`, then it is easy to change the configuration of the listener on mylsnr2.mycompany.com. Edit the file and change the lines `ServerName` and `Port` in the `ORACLE_HOME/Apache/Apache/conf/httpd.conf` file for the second listener as follows:

```
ServerName mylsnr1.mycompany.com (was mylsnr2.mycompany.com)
Port 7777                        (was 8888)
```

   - If the URLs are being generated using the CGI environment variable `HTTP_HOST`, you need to override the CGI environment variables for the Oracle Application Server listener running on Port 8888. Add the following lines to the `ORACLE_HOME/Apache/modplsql/conf/dads.conf` file for each DAD to override the default CGI environment variables `HOST`, `SERVER_NAME`, and `SERVER_PORT`:

```
PlsqlCGIEnvironmentList  SERVER_NAME  mylsnr1.mycompany.com
PlsqlCGIEnvironmentList  SERVER_PORT  7777
PlsqlCGIEnvironmentList  HOST mylsnr1.us.oracle.com:7777
```

In all cases, the intent is to fool the application to generate URLs as if there never was a second listener.

3. Test the setup and make sure that you can access all the DADs without any problems.
4. In this setup, the main listener mylsnr1.mycompany.com can be configured based on the total load on the Oracle Application Server listener. The second listener on mylsnr2.mycompany.com can be fine-tuned based on just the `mod_plsql` requests being made.

Overhead Problems

While executing some of the Portal stored procedures, `mod_plsql` may incur a `Describe` overhead which would result in two extra round trips to the database for a successful execution. This has performance implications.

See Also: *Oracle Application Server 10g mod_plsql User's Guide*

The Describe Overhead

In order to execute PL/SQL procedures, `mod_plsql` needs to know about the datatype of the parameters being passed in. Based on this information, `mod_plsql` binds each parameter either as an array or as a scalar. One way to know the procedure signature is to describe the procedure before executing it. However, this approach is not efficient because every procedure has to be described before execution. To avoid the describe overhead, `mod_plsql` looks at the number of parameters passed for each parameter name. It uses this information to assume the datatype of each variable. The logic is simply that if there is a single value being passed, then the parameter is a scalar, otherwise it is an array. This works for most cases but fails if there is an attempt to pass a single value for an array parameter or pass multiple values for a scalar. In such cases, the first attempt to execute the PL/SQL procedure fails. `mod_plsql` issues a `Describe` call to get the signature of the PL/SQL procedure and binds each parameter based on the information retrieved from the `Describe` operation. The procedure is re-executed and results are sent back.

This `Describe` call occurs transparently to the procedure, but internally `mod_plsql` has encountered two extra round trips, one for the failed execute call and the other for the describe call.

Avoiding the Describe Overhead

You can avoid performance problems with the following:

- Use flexible parameter passing.
- Always ensure that you pass multiple values for arrays. For single values, you can pass dummy values which are ignored by the procedure.
- Use the following workaround which defines a two-parameter style procedure which defaults the unused variables.

1. Define a scalar equivalent of your procedure which internally calls the original procedure. For example, the original package could be similar to the following example:

```
CREATE OR REPLACE PACKAGE testpkg AS
 TYPE myArrayType is TABLE of VARCHAR2(32767) INDEX BY binary_ integer;
 PROCEDURE arrayproc (arr myArrayType);
END testpkg;
/
```

2. If you are making URL calls like `/pls/.../testpkg.arrayproc?arr= 1`, change the specification to be similar to the following:

```
CREATE OR REPLACE PACKAGE testpkg AS
 TYPE myArrayType is TABLE of VARCHAR2( 32767) INDEX BY binary_integer;
  PROCEDURE arrayproc (arr varchar2);
  PROCEDURE arrayproc (arr myArrayType);
END testpkg;
/
```

3. The procedure `arrayproc` should be similar to:

```
CREATE OR REPLACE PACKAGE BODY testpkg AS
PROCEDURE arrayproc (arr varchar2) IS
  localArr myArrayType;
BEGIN
  localArr( 1) := arr;
  arrayproc (localArr);
END arrayproc;
```

The Flexible Parameter Passing (four-parameter) Overhead

Round-trip overhead exists if a PL/SQL procedure is using the older style four-parameter interface. The PL/SQL Gateway first tries to execute the procedure by using the two-parameter interface. If this fails, the PL/SQL Gateway tries the four-parameter interface. This implies that all four-parameter interface procedures experience one extra round-trip for execution.

- Avoiding the flexible parameter passing overhead

  To avoid this overhead, it is recommended that you write corresponding wrappers that use the two-parameter interface and internally call the four-parameter interface procedures. Another option is to change the specification of the original procedure to default to the parameters that are not passed in the two-parameter interface. The four-parameter interface has been provided only for backward compatibility and will be deprecated in the future.

- Using flexible parameters and the exclamation mark

  The flexible parameter passing mode in Oracle Application Server expects the PL/SQL procedure to have the exclamation mark before the procedure name. Due to performance implications of the auto-detect method used in Oracle Application Server, the exclamation mark is now required for flexible parameter passing in Oracle Application Server. In Oracle Application Server, each procedure is described completely before being executed. The procedure `Describe` call determines the signature of the procedure and requires around-trip to the database. The PL/SQL Gateway in Oracle Application Server avoids this round trip by having end-users explicitly indicate the flexible parameter passing convention by adding the exclamation mark before the procedure.

Using Caching with PL/SQL Based Web Applications

Caching can improve the performance of PL/SQL based Web applications. To improve performance, you can cache Web content generated by PL/SQL procedures in the middle-tier and decrease the database workload.

This section covers the techniques used in caching, including the following:

- Using the Validation Technique - An application asks the server if the page has been modified since it was last presented.
- Using the Expires Technique - Based upon a specific time period, the PL/SQL based Web application determines the page will be cached, or should be generated again.

- System- and User-level Caching with PL/SQL Based Web Applications - This is valid whether you are using the Validation Technique or the Expires Technique. The level of caching is determined by whether a page is cached for a particular user or for every user in the system.

These techniques and levels are implemented using `ows_cache` packages located inside the PL/SQL Web Toolkit.

> **See Also:** *Oracle Application Server 10g PL/SQL Web Toolkit Reference*

Using the Validation Technique

In general, the validation technique basically asks the server if the page has been modified since it was last presented. If it has not been modified, the cached page will be presented to the user. If the page has been modified, a new copy will be retrieved, presented to the user and then cached.

There are two methods which use the Validation Technique, Last-Modified method and the Entity Tag method. The next two sections show how these techniques are used in the HTTP protocol. Although the PL/SQL Gateway does not use the HTTP protocol, many of the same are principles are used.

Last-Modified

When a Web page is generated using the HTTP protocol, it contains a **Last-Modified** Response Header. This header indicates the date, relative to the server, of the content that was requested. Browsers save this date information along with the content. When subsequent requests are made for the URL of the Web page, the browser then:

1. Determines if it has a cached version.
2. Extracts the date information.
3. Generates the Request Header **If-Modified-Since**.
4. Sends the request the server.

Cache-enabled servers look for the **If-Modified-Since** header and compare it to the date of their content. If the two match, an HTTP Response status header such as "HTTP/1.1 304 Not Modified" is generated, and no content is streamed. After receiving this status code, the browser can reuse its cache entry because it has been validated.

If the two do not match, an HTTP Response header such as "HTTP/1.1 200 OK" is generated and the new content is streamed, along with a new **Last-Modified Response h**eader. Upon receipt of this status code, the browser must replace its cache entry with the new content and new date information.

Entity Tag Method

Another validation method provided by the HTTP protocol is the **ETag** (Entity Tag) Response and Request header. The value of this header is a string that is opaque to the browser. Servers generate this string based on their type of application. This is a more generic validation method than the **If-Modified-Since** header, which can only contain a date value.

The **ETag** method works very similar to the Last Modified method. Servers generate the ETag as part of the Response Header. The browser stores this opaque header value along with the content that is steamed back. When the next request for this content arrives, the browser passes the **If-Match** header with the opaque value that it stored to the server. Because the server generated this opaque value, it is able to determine what to send back to the browser. The rest is exactly like the **Last-Modified** validation method as described above.

Using the Validation Technique for mod_plsql

Using HTTP validation caching as a framework, the following is the Validation Model for `mod_plsql`.

PL/SQL based Web applications that want to control the content being served should use this type of caching. This technique offers some moderate performance gains. One example of this would be a Web application that serves dynamic content that can change at any given time. In this case, the Web application needs full control over what is being served. Validation caching always asks the Web application whether the cached content is stale or not before serving it back to the browser.

Figure 8–1 shows the use of the validation technique for `mod_plsql`.

1. The Oracle HTTP Server receives a PL/SQL procedure request from a client server. The Oracle HTTP Server routes the request to `mod_plsql`.
2. `mod_plsql` prepares the request.
3. `mod_plsql` invokes the PL/SQL procedure in the Web application. `mod_plsql` passes the usual Common Gateway Interface (CGI) environment variables to the Web application.

4. The PL/SQL procedure generates content to pass back. If the PL/SQL procedure decides that the generated content is cacheable, it calls the `owa_cache` procedure from the PL/SQL Web Toolkit to set the tag and cache level:

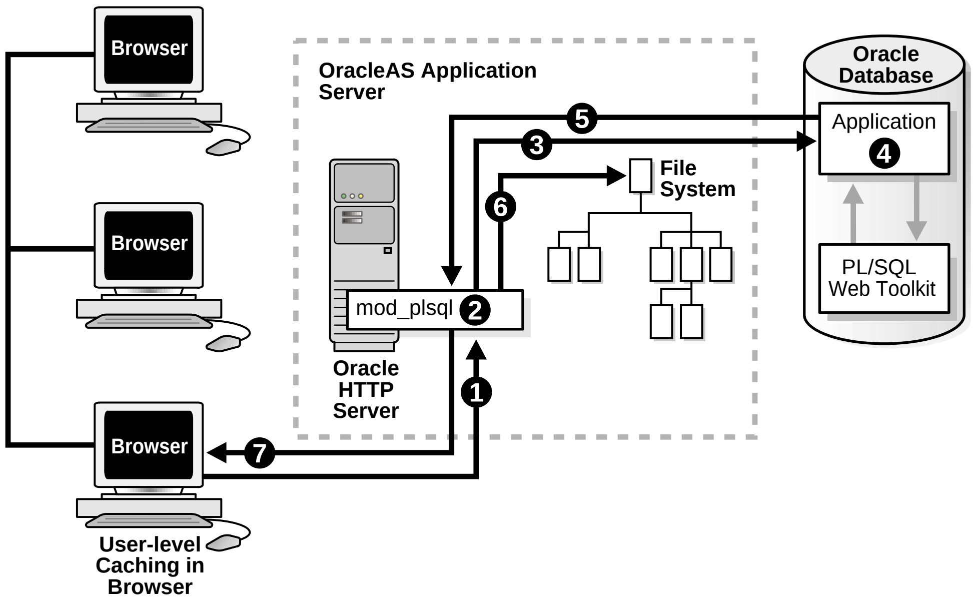
```
owa_cache.set_cache(p_etag, p_level);
```

Table 8–3 Validation Model Parameters

Parameter	Description
`set_cache` procedure	Sets up the headers to notify `mod_plsql` that the content being streamed back can be cached. Then, the `mod_plsql` caches the content on the local file system along with the tag and caching level information as it is streamed back to the browser.
`p_etag`	The string that the procedure generates to tag the content.
`p_level`	The caching level: `SYSTEM` for system level or `USER` for user level.

5. The HTML is returned to `mod_plsql`.
6. `mod_plsql` stores the cacheable content in its file system for the next request.
7. The Oracle HTTP Server sends the response to the client browser.

Figure 8–1 Validation Technique



Second Request Using the Validation Technique

Using the Validation Technique for `mod_plsql`, a second request is made by the client browser for the same PL/SQL procedure.

Figure 8–2 shows the second request using the Validation Technique.

1. `mod_plsql` detects that it has a cached content for the request.
2. `mod_plsql` forwards the same tag and caching level information (from the first request) to the PL/SQL procedure as part of the CGI environment variables.
3. The PL/SQL procedure uses these caching CGI environment variables to check if the content has changed. It does so by calling the following `owa_cache` functions from the PL/SQL Web Toolkit:

   ```
   owa_cache.get_etag;
   owa_cache.get_level;
   ```

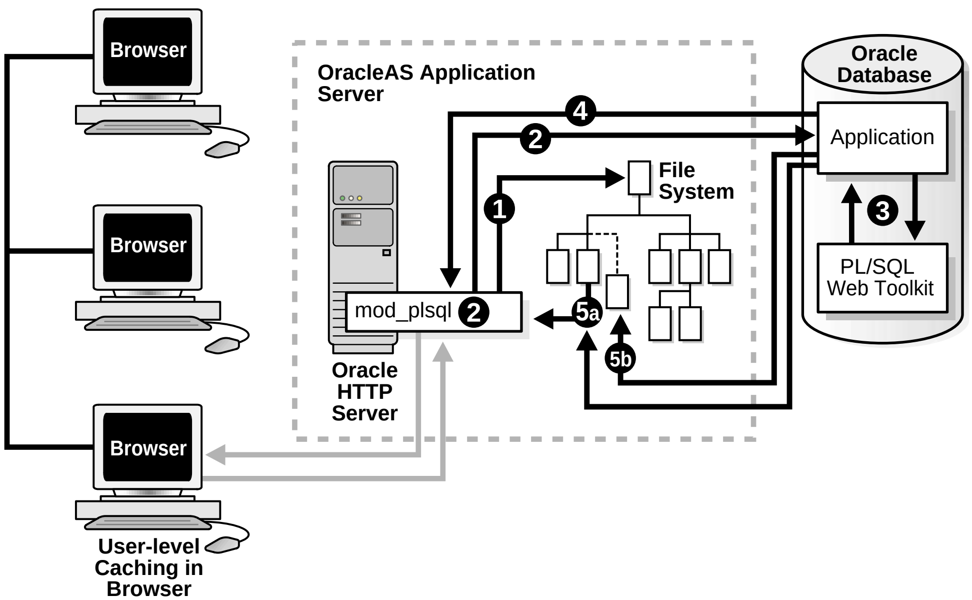
   These owa functions get the tag and caching level.
4. The Web application sends the caching information to `mod_plsql`.

5. Based on that information determines whether the content needs to be regenerated or can be served from the cache.

   a. If the content is still the same, the procedure calls the `owa_cache.set_not_modified` procedure and generates no content. This causes `mod_plsql` to use its cached content. The cached content is directly streamed back to the browser.

   b. If the content has changed, it generates the new content along with a new tag and caching level. `mod_plsql` replaces its stale cached copy with a new one and updates the tag and caching level information. The newly generated content is streamed back to the browser.

Figure 8–2 Validation Technique-Second Request



Using the Expires Technique

In the validation model, mod_plsql always asks the PL/SQL procedure if it can serve the content from the cache. In the expires model, the procedure preestablishes the content validity period. Therefore, mod_plsql can serve the content from its cache without asking the procedure. This further improves performance because no interaction with the database is required.

This caching technique offers the best performance. Use if your PL/SQL based Web application is not sensitive to serving stale content. One example of this is an application that generates news daily. The news can be set to be valid for 24 hours. Within the 24 hours, the cached content is served back without contacting the application. This is essentially the same as serving a file. After 24 hours, mod_plsql will again fetch new content from the application.

Assume the same scenario described for the Validation model, except the procedure uses the Expires model for caching.

Figure 8–3 shows the use of the expires technique for mod_plsql.

1. The Oracle HTTP Server receives a PL/SQL Server Page request from a client server. The Oracle HTTP Server routes the request to mod_plsql.
2. The request is forwarded by mod_plsql to the Oracle Database.
3. mod_plsql invokes the PL/SQL procedure in the application and passes the usual Common Gateway Interface (CGI) environment variables to the application.
4. The PL/SQL procedure generates content to pass back. If the PL/SQL procedure decides that the generated content is cacheable, it calls the owa_cache procedure from the PL/SQL Web Toolkit to set the validity period and cache level:

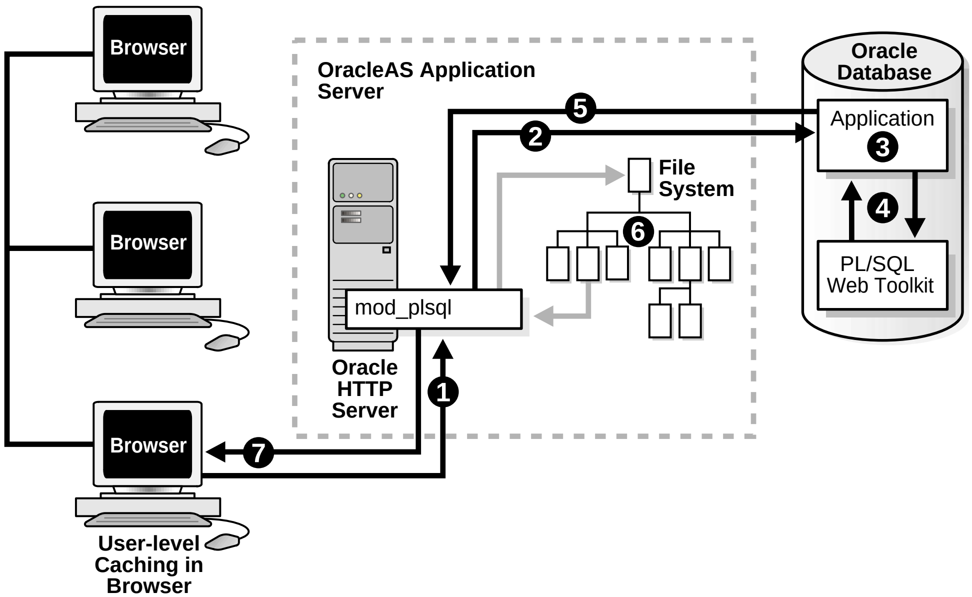
```
owa_cache.set_expires(p_expires, p_level);
```

Table 8–4 Expires Model Parameters

Parameter	Description
set_expires procedure	Sets up the headers to notify mod_plsql that Expires caching is being used. mod_plsql then caches the content to the file system along with the validity period and caching level information.
p_expires	Number of minutes that the content is valid.
p_level	Caching level.

5. The HTML is returned to `mod_plsql`.
6. `mod_plsql` stores the cacheable content in its file system for the next request.
7. The Oracle HTTP Server sends the response to the client browser.

Figure 8–3 The Expires Technique



Second Request Using the Expires Technique

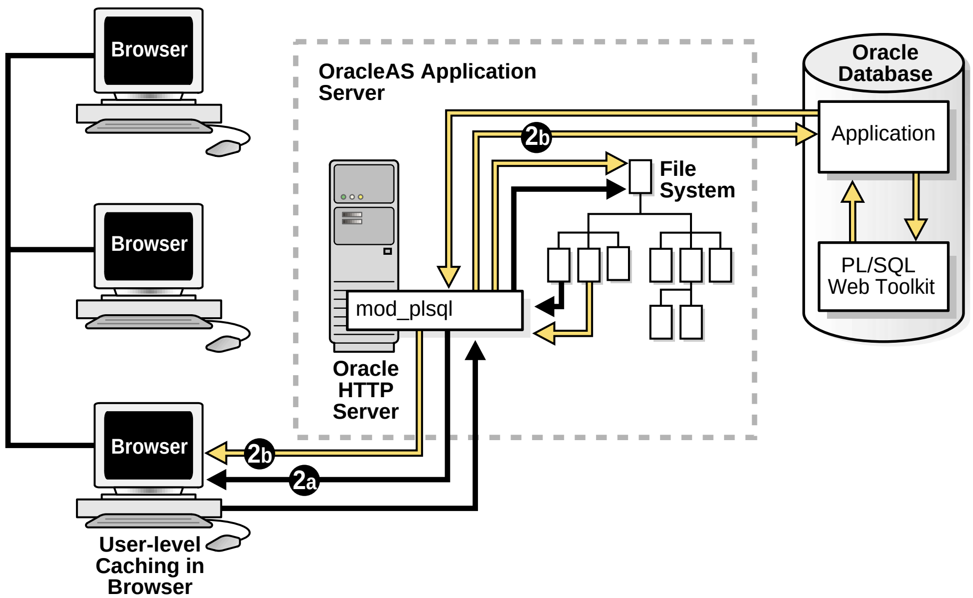
Using the same expires model explained above, a second request is made by the client browser for the same PL/SQL procedure.

Figure 8–4 shows the second request using the Expires Technique.

1. `mod_plsql` detects that it has a cached copy of the content that is expires-based.
2. `mod_plsql` checks the content's validity by taking the difference between the current time and the time this cache file was created.
   a. If this difference is within the validity period, the cached copy is still fresh and will be used without any database interaction. The cached content is directly streamed back to the browser.

b. If the difference is not within the validity period, the cached copy is stale. `mod_plsql` invokes the PL/SQL procedure and generates new content. The procedure then decides whether to use expires-based caching again. If so, it also determines the validating period for this new content. The newly generated content is streamed back to the browser.

Figure 8–4 The Expires Technique-Second Request



System- and User-level Caching with PL/SQL Based Web Applications

A PL/SQL procedure determines whether generated content is system-level content or user-level. This helps the PL/SQL Gateway cache to store less redundant files if more than one user is looking at the same content. It decides this by:

- For **system-level** content, the procedure passes the string `SYSTEM` as the caching level parameter to the `owa_cache` functions (`set_cache` for validation model or `set_expires` for expires model). This is for every user that shares the cache.

  By using system-level caching, you can save both space in your file system and time for all users in the system. One example of this would be a Web application that generates content that is intended for everybody using the Web application. By caching the content with the system-level setting, only one copy of the content is cached in the file system. Furthermore, every user on that system benefits since the content is served directory from the cache.

- For **user-level** content, it passes the string USER as the parameter for the caching level. This is for a specific user that is logged in. The stored cache is unique for that user. Only that user can use the cache.The type of user is determined by the authentication mode. Refer to the following table for the different types of users.

Table 8–5 Type of User Determined by Authentication Mode

Authentication Mode	Type of User
Single Sign On (SSO)	Lightweight user
Basic	Database user
Custom	Remote user

For example, if no user customizes a PL/SQL based Web application, then the output can be stored in a system-level cache. There will be only one cache copy for every user on the system. User information is not used since the cache can be used by multiple users.

However, if a user customizes the application, a user-level cache is stored for that user only. All other users still use the system level cache. For a user-level cache hit, the user information is a criteria. A user-level cache always overrides a system-level cache.

> **See Also:** *Oracle Application Server 10g mod_plsql User's Guide* for more information on Authentication Mode.

PL/SQL Web Toolkit functions (owa_cache package)

Your decision whether to use the Validation technique or the Expires technique determines which `owa_cache` functions to call.

The `owa_cache` package contains procedures to set and get special caching headers and environment variables. These allow developers to use the PL/SQL Gateway cache more easily. This package should already be installed in your database.

Table 8–6 lists the primary functions to call.

Table 8–6 Primary owa_cache Functions

owa Functions	Purpose
`owa_cache.set_cache (p_etag IN varchar2, p_level IN varchar2)`	Validation Model - Sets up the headers. ▪ `p_etag` parameter tags the generated content. ▪ `p_level` parameter is the caching level to use.
`owa_cache.set_not_modified`	Validation Model - Sets up the headers to notify `mod_plsql` to use the cached content. Only used when a validation -based cache hit occurs.
`owa_cache.get_level`	Validation Model - Gets the caching level, `USER` or `SYSTEM`. Returns null if the cache is not hit.
`owa_cache.get_etag`	Validation Model - Gets the tag associated with the cached content. Returns null if the cache is not hit.
`owa_cache.set_expires( p_expires IN number,` p_level IN varchar2)	Expires Model - Sets up the headers. ▪ `p_expires` parameter is the number of minutes the content is valid. ▪ `p_level` parameter is the caching level to use.

Tuning File System Cache to Improve Caching Performance

You can configure and use a File System Cache to improve the performance of OracleAS Portal applications and generic PL/SQL based Web applications.

This section covers the following topics:

- Introducing File System Cache Tuning
- Enabling File System Cache
- Configuring File System Cache to Reside on a Faster File System
- Configuring Session Cache Using Performant File System (OracleAS Portal Only)
- Resizing File System Cache
- Configuring Cache Cleanup

Introducing File System Cache Tuning

This section covers mod_plsql related File System Cache tuning options. Cache contents are cached using Operating System supplied file system calls; the cached contents are not stored in the mod_plsql memory space. Using the mod_plsql File System Cache, the contents of the cache may be in memory when the Operating System supports, and the system is configured to use features such as memory disk (some UNIX platforms support memory disk based fast storage).

The information in this section can improve the performance of PL/SQL based Web applications when mod_plsql is configured to use the File System Cache. For example, OracleAS Portal uses the File System Cache, and therefore, OracleAS Portal performance should improve when the File System Cache is properly tuned.

Table 8–7 lists the cache related parameters that you can set for mod_plsql. Set these parameters in the cache.conf file that is available on UNIX in the directory, $ORACLE_HOME/Apache/modplsql/conf, and on Windows, this is found in the directory, %ORACLE_HOME%\Apache\modplsql\conf.

Note: The file cache.README in the conf directory includes a full description of each parameter, and provides examples showing how to set parameter values.

Table 8–7 mod_plsql cache.conf Configuration Parameter Summary

Parameter	Description
`PlsqlCacheCleanupTime`	Sets the interval for running cache cleanup routines. Default for Oracle Application Server installation: `Everyday 23:00` (run cleanup routine daily at 11PM local time) See Also: "Configuring Cache Cleanup" on page 8-38
`PlsqlCacheDirectory`	Defines the directory that holds the `mod_plsql` cache. Default Value for Oracle Application Server installation: On UNIX systems, the default directory for the error log is: . `$ORACLE_HOME/Apache/modplsql/cache` On Windows systems, the default directory is: `%ORACLE_HOME%\Apache\modplsql\cache` See Also: "Configuring File System Cache to Reside on a Faster File System" on page 8-31
`PlsqlCacheEnable`	Enables the file system cache. Default for Oracle Application Server installation: `On` See Also: "Enabling File System Cache" on page 8-31
`PlsqlCacheMaxAge`	Controls the aging, in days for the cache contents. Default for Oracle Application Server installation: `30` (days) See Also: "Setting the Days of Aging for Cache With PlsqlCacheMaxAge" on page 8-37
`PlsqlCacheMaxSize`	Sets the maximum size, in bytes, for an individual file stored in the cache. Default for Oracle Application Server installation: 1048576 (1 Megabyte) See Also: "Setting the Maximum File Size for a Cache File With PlsqlCacheMaxSize" on page 8-37
`PlsqlCacheTotalSize`	Limits the total size of the cache. The value is specified in bytes. Default `for Oracle Application Server installation:` 20971520 (20 Megabytes) See Also: "Resizing File System Cache" on page 8-35

Enabling File System Cache

The `cache.conf` parameter `PlsqlCacheEnable` enables `mod_plsql` caching. For maximum performance, enable `PlsqlCacheEnable` by setting the value of this parameter to `On`.

> **Note:** Only applications that support PLSQL caching, such as Oracle Portal, will benefit by setting `PlsqlCacheEnable` to `On`.

Configuring File System Cache to Reside on a Faster File System

This section describes how to configure a File System Cache to reside on a separate disk. When you use File System Cache and store the cache on a faster separate disk, performance should improve for all types of Web applications using File System Cache, including OracleAS Portal and generic PL/SQL based Web applications.

When you configure File System Cache, the cache can reside either on a separate physical disk or in a memory disk.

To set up a File System Cache on a separate disk:

1. Assume that the file system for the cache resides at the location:

   On UNIX: `/u01/cache`

   On Windows: `E:\cache`

   Update the file:

   On UNIX: `$ORACLE_HOME/Apache/modplsql/conf/cache.conf`

   On Windows: `%ORACLE_HOME%\Apache\modplsql\conf\cache.conf`

   Change the cache parameter `PlsqlCacheDirectory`:

   On UNIX: `PlsqlCacheDirectory /u01/cache`

   On Windows: `PlsqlCacheDirectory  E:\cache`

> **Note:** When you edit the `cache.conf` file, use the following command to update the Oracle Application Server configuration:
>
> ```
> $ORACLE_HOME/dcm/bin/dcmctl updateConfig -ct ohs
> ```

2. This step is only required when the Oracle Application Server mid-tier runs OracleAS Portal (this only applies to OracleAS Portal).

   Open the file:

   On UNIX:

   `$ORACLE_HOME/j2ee/OC4J_Portal/applications/portal/portal/WEB-INF/web.xml`

   On Windows:
   `%ORACLE_HOME\j2ee\OC4J_Portal\applications\portal\portal\WEB-INF\web.xml`

   And add the following XML fragment to the init parameters for the page servlet:

   On UNIX:

```
<servlet>
  <servlet-name>page</servlet-name>
  <servlet-class>oracle.webdb.page.ParallelServlet</servlet-class>
  <init-param>
  <param-name>cacheDir</param-name>
  <param-value>/u01/cache</param-value>
  </init-param>
.
.
.
</servlet>
```

   On NT:

```
<servlet>
  <servlet-name>page</servlet-name>
  <servlet-class>oracle.webdb.page.ParallelServlet</servlet-class>
 <init-param>
 <param-name>cacheDir</param-name>
  <param-value>E:\cache</param-value>
  </init-param>
.
.
.
</servlet>
```

3. Restart Oracle Application Server for the configuration changes to take effect.

Configuring Session Cache Using Performant File System (OracleAS Portal Only)

This section provides tips and techniques for reducing the time required to access and login to OracleAS Portal.

Note:

- The information in this section only applies only to OracleAS Portal, and only when OracleAS Portal is configured in the mid-tier.
- The information in this section only applies on UNIX systems.

Given that each OracleAS Portal session cache item is small, accessed per-request, and is volatile, you can improve the session cache performance by ensuring that the session cache resides on a performant file system.

Options for a more performant file system include memory-based file systems that are available on many UNIX platforms.

OracleAS Portal uses session cookies to maintain session details for each OracleAS Portal user. The session cookie is encrypted and contains important information, including: the database username, the lightweight username, and the Globalization Support characteristics of the session. In order for `mod_plsql` to execute a OracleAS Portal request, it needs to get hold of the database username from the session cookie. To avoid performing an expensive decrypt operation with each user request, `mod_plsql` decrypts the session cookie once and maintains the relevant details in a OracleAS Portal session cache that is stored on the local file system.

Usually, the OracleAS Portal session cache directory resides under `$ORACLE_HOME/Apache/modplsql/cache` on UNIX, and you specify OracleAS Portal session cache configuration in the configuration file `$ORACLE_HOME/Apache/modplsql/conf/cache.conf` on UNIX

To configure the OracleAS Portal session cache directory to use a performant file system, do the following:

1. Determine how much space the OracleAS Portal session cache needs. For example:
   - 10,000 public session cookies will be generated - one for each public user.
   - 5,000 Portal session cookies will be generated - one for each user logged into OracleAS Portal

- 5,000 public SSO session cookies will be generated - one for each public user.
- 5,000 SSO session cookies will be generated – one for each logged in SSO user.
- Each session cookie item is roughly 400 bytes each.

For this example, the total OracleAS Portal session cache size required is:

```
(Total Session Cookies * size of each cookie) =
(10,000 + 5,000 + 5,000*2) *400 bytes =
10 Megabytes
```

Caution: This calculation only applies for session cookies that get created in one day. The total OracleAS Portal session cache calculation does not account for session cache items that got created earlier on.

There is no configuration parameter to specify how much space to use for the OracleAS Portal session cache. The session caching logic will keep writing to the session cache directory and assume that the session cache cleanup is done frequently enough to make sure that the disk never runs out of space. If you are using your swap space for the session cache, it is extremely important that you never run out of swap space on the system. If the disk does run out of space, session caching will not occur, but OracleAS Portal will continue to run with a substantial degradation in performance. This will adversely affect the system in general and should be monitored

---

2. Identify a file system with enough space to hold the OracleAS Portal session cache; then, create a session cache directory.

   If you want to use the swap space in `/tmp` for the OracleAS Portal session cache, you need to do the following

   - Login as the user that will be running the Apache listener and the OC4J instances.
   - Create a new directory for the OracleAS Portal session cache. For example, using the command:

     ```
     % mkdir /tmp/session
     ```

- Revoke all privileges for everyone else on the new directory:

  ```
  % chmod 700 /tmp/session
  ```

> **Caution:** No other component in the system should be creating files under the OracleAS Portal session cache directory. Make sure that you create a new directory for the session cache.

3. Remove the existing OracleAS Portal session cache directory `$ORACLE_HOME/Apache/modplsql/cache/session` (applies on UNIX only).

   It is safe to delete all files under the OracleAS Portal session cache directory since these are cached items which get re-created as users access OracleAS Portal:

   ```
   % rm -rf $ORACLE_HOME/Apache/modplsql/cache/session
   ```

4. Setup the OracleAS Portal session cache directory to point to the new file system by creating a symbolic link:

   ```
   %cd $ORACLE_HOME/Apache/modplsql/cache
   % ln -s /tmp/session session
   ```

Resizing File System Cache

This section covers the following topics:

- Setting the Total Cache Size With PlsqlCacheTotalSize
- Setting the Days of Aging for Cache With PlsqlCacheMaxAge
- Setting the Maximum File Size for a Cache File With PlsqlCacheMaxSize

Setting the Total Cache Size With PlsqlCacheTotalSize

The default Oracle Application Server installation sets the cache size to 2097152 bytes (20 Megabytes). This setting is suitable for small to medium Oracle Application Server deployments. The default size is not suitable for large Oracle Application Server deployments.

To control the cache size, set the PlsqlCacheTotalSize parameter in the file cache.conf. On UNIX systems, this file is located under $ORACLE_HOME/Apache/modplsql/conf directory. On Windows systems, this file is located under %ORACLE_HOME%\Apache\modplsql\conf.

You need to set the cache size high enough to achieve a high cache hit ratio. Try to set the cache size large enough so that frequently accessed content stays cached. It is also important to limit the amount of disk space, so that the cache size does not grow too large. Correct tuning for the cache size provides enough cache to hold all frequently accessed content while preventing the cache size from growing too large, since a very large cache is inefficient to search.

The value for PlsqlCacheTotalSize is specified as a number of bytes. 1MB equals 1048576 bytes. This setting is a soft limit on the amount of cache allocated. In some cases, the cache size may grow beyond this limit until the next cleanup operation. Therefore, the hard limit on the cache size is the underlying physical hard disk size. When this limit is reached, no cache content can be written out to disk until space is available.

To determine a reasonable cache size, do the following:

1. Turn on mod_plsql performance logging by setting the LogLevel in httpd.conf to the info level to enable mod_plsql logging.
2. Monitor the error_log on a daily basis. On UNIX systems, the default directory for the error log is: $ORACLE_HOME/Apache/Apache/log. On Windows systems, the default directory is: %ORACLE_HOME%\Apache\Apache\log.

The mod_plsql error_log entries have the form:

```
[info] mod_plsql: cachecleanup deleted=2571 max_age=96,2178852b
kept=1042,25585368b time=128s limit=25600000b
```

where:

deleted is the number of cache files that got deleted during the cleanup process.

max_age is the number of cache files and total size that got deleted because they haven't been used for some time.

kept is the number of cache files and total size that was kept after the cleanup process.

time is the amount of time to perform the cleanup.

`limit` is the total cache size. This is the value of the `PlsqlCacheTotalSize` setting.

Interpret the entries in the error log as follows:

- If a high number of files are being deleted when compared to the number of files that were kept, this is a clear indication that your cache size is too small. You probably need to increase the size of the cache.
- If a low number of files being deleted when compared to the number of files that were kept is observed, this is an indication that your cache size is probably too big. If you have enough disk space, you can chose to leave it as it or you can decrease the size of the cache to reclaim some disk space.

> **See Also:** *Oracle Application Server 10g mod_plsql User's Guide*

Setting the Days of Aging for Cache With PlsqlCacheMaxAge

Using the `PlsqlCacheMaxAge` parameter, you can control the "staleness" of cache content The value for parameter is specified in units of days. The default Oracle Application Server installation sets this parameter to 30 (days). This means cache contents are kept in the cache if they are less than 30 days old. After 30 days the contents are considered for deletion during the cleanup process.

The `max_age` information in `mod_plsql error_log` shows cache file aging information. If your site is a highly dynamic site, it would make sense to configure this setting to a lower value, since the older cache content will usually not be used again and, therefore, the lower value does not affect the cache hit ratio. If the site contains many static pages, it would make sense to increase the value of `PlsqlCacheMaxAge` so that the cleanup process does not deliberately delete the cache content.

Setting the Maximum File Size for a Cache File With PlsqlCacheMaxSize

Using the `PlsqlCacheMaxSize` parameter, you can specify the maximum size for individual files in the cache. Using this parameter prevents the case in which one cache file fills up the entire cache.

The default Oracle Application Server installation sets this parameter to the value 1048576 (bytes). In general, set this parameter to a value that represents about 1-3% of the total cache size.

Configuring Cache Cleanup

The cache cleanup parameter determines the frequency in which the File System Cache is examined and, if necessary, cleaned up. The cache cleanup parameter, `PlsqlCacheCleanupTime` is specified in the `cache.conf` file. The frequency can be set to daily, weekly, or monthly. When specifying weekly cleanup, it is possible to specify the day of the week and the time of the day.

The default Oracle Application Server installation sets the `PlsqlCacheCleanupTime` to daily at 11PM local time. Therefore, by default, every night at 11PM, the cleanup routine runs. When you select the monthly frequency, the cleanup occurs on the first Saturday of each month.

Configuring this parameter correctly is important since cleaning up too often can result in a lower cache hit ratio and when cleaning does not occur often enough, the cache's disk usage may be excessive.

Monitor the cleanup activities using the entries in the `mod_plsql` `error_log`; then tune the cleanup parameter, `PlsqlCacheCleanupTime` by analyzing the entries.

```
[info] mod_plsql: cachecleanup deleted=2571 max_age=96,2178852b
kept=1042,25585368b time=128s limit=25600000b
```

Note the following:

- Seeing a large number for the cleanup time can be an indication that the cleanup frequency is set too low. When the log indicates that the cleanup operation is busy examining or deleting many cache files, increasing the cleanup frequency should decrease the time spent in the cleanup operation.
- If a high number files are being deleted during the cleanup operation because of "staleness", this is an indication that the cleanup frequency is too low. In this case, increase the frequency so that the cleanup can actively delete "stale" cache content more frequently.

Oracle HTTP Server Directives

To improve PL/SQL performance in Oracle Application Server, you need to tune the Oracle HTTP Server directives appropriately for your configuration.

See Also:

- "Configuring Oracle HTTP Server Directives" on page 5-11
- *Oracle HTTP Server Administrator's Guide*.Chapter 3, "Managing Server Processes" and Chapter 4, "Managing the Network Connection"

9

Instrumenting Applications With DMS

The Oracle Dynamic Monitoring Service (DMS) enables application developers, support analysts, system administrators, and others to measure application specific performance information. This chapter describes DMS and shows a sample application that demonstrates how to instrument Oracle Application Server Java applications using DMS.

Note: Oracle Application Server provides a number of built-in metrics. Using DMS to instrument applications adds new metrics to the set of built-in metrics.

This chapter covers the following topics:

- Introducing DMS Performance Metrics
- Adding DMS Instrumentation To Java Applications
- Validating and Testing Applications Using DMS Metrics
- Understanding DMS Security Considerations
- Advanced DMS Topics

See Also: Appendix A, "Performance Metrics"

Introducing DMS Performance Metrics

The Dynamic Monitoring Service (DMS) API allows you to add performance instrumentation to Oracle Application Server applications. During runtime DMS collects performance information, called DMS metrics, that developers, system administrators, and support analysts use to help analyze system performance or monitor system status.

This section covers the following topics:

- Instrumenting Applications With DMS Metrics
- Monitoring DMS Metrics
- Understanding DMS Terminology (Nouns and Sensors)
- Isolating Expensive Intervals Using PhaseEvent Metrics

Note: Oracle Application Server components, including OC4J, provide a number of predefined metrics. For a listing of the predefined metrics see Appendix A, "Performance Metrics".

Instrumenting Applications With DMS Metrics

DMS **Instrumentation** refers to the process of inserting DMS calls into application code. Using the DMS API is a simple and efficient way to enable your application to measure, collect, and save performance information.

To create DMS metrics developers add calls that notify DMS when events occur, when important intervals begin and end, or when pre-computed values change their state. At runtime, DMS stores metrics in memory and allows you to save or view the metrics.

Oracle Application Server includes built-in DMS metrics. By adding DMS calls to your applications, you expand the set of built-in metrics. When you instrument your applications with DMS calls, you use the same API that the built-in metrics use. In addition, to save and display your metrics, use the same monitoring tools that you use with built-in metrics.

See Also: "Adding DMS Instrumentation To Java Applications" on page 9-8

Monitoring DMS Metrics

Monitoring DMS metrics refers to the process of retrieving performance metrics. When an application runs, DMS stores metrics in memory and allows you to show metrics on the console, save metrics to a file, or to view metrics using a web browser.

Oracle Application Server provides several runtime tools for viewing and saving DMS metrics, including `dmstool`, and the `Spy` and `AggreSpy` Servlets.

Example 9–1 shows a set of metrics output using `dmstool`.

Example 9–1 Set of Sample dmsDemo Metrics Using dmstool

```
/dmsDemo [type=n/a]
  /dmsDemo/BasicBinomial [type=MathSeries]
   computeSeries.active:        0       threads
   computeSeries.avg:         220.333   msecs
   computeSeries.completed:     3       ops
   computeSeries.maxActive:     1       threads
   computeSeries.maxTime:     435       msecs
   computeSeries.minTime:      35       msecs
   computeSeries.time:  661     msecs
   lastComputed.value:  100891344545564193334812497256
   loops.count: 147     ops
```

See Also: Chapter 2, "Monitoring Oracle Application Server"

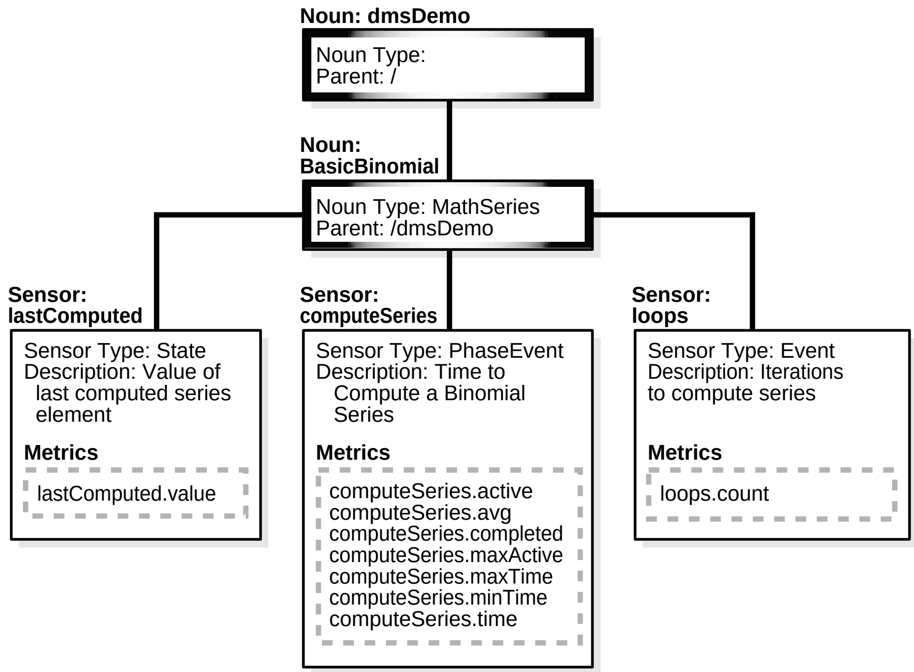
Understanding DMS Terminology (Nouns and Sensors)

This section introduces the terminology you need to understand to use DMS. Figure 9–1 illustrates the organization of a set of DMS metrics corresponding to the metrics in the demo application described in this chapter and the metrics shown in Example 9–1.

This section covers the following topics:

- DMS Metrics
- DMS Sensors
- DMS Nouns
- DMS Object Relationships

Figure 9–1 Organization of Sample Metrics From dmsDemo Application



DMS Metrics

DMS Metrics track performance information that developers, system administrators, and support analysts use to help analyze system performance or monitor system status.

DMS Sensors

DMS Sensors measure performance data and allow DMS to define and collect a set of metrics. Certain metrics are always included with a Sensor and other metrics are optionally included with a Sensor.

DMS PhaseEvent Sensors A DMS **PhaseEvent Sensor** measures the time spent in a specific section of code that has a beginning and an end. Use a PhaseEvent Sensor to track time in a method or in a block of code.

DMS can calculate optional metrics associated with a PhaseEvent, including: the average, maximum, and minimum time that is spent in the PhaseEvent.

Table 9–1 describes metrics available with a PhaseEvent.

Table 9–1 DMS PhaseEvent Sensor Metrics

Metric	Description
sensor_name.time	Specifies the total time spent in the phase *sensor_name*. Default: time is a default PhaseEvent Sensor.
sensor_name.completed	Specifies the number of times the phase *sensor_name*, has completed since the process was started.
sensor_name.minTime	Specifies the minimum time spent in the phase *sensor_name*, for all the times the phase completed.
sensor_name.maxTime	Specifies the maximum time spent in the phase *sensor_name*, over all the times the *sensor_name* phase completed.
sensor_name.avg	Specifies the average time spent in the phase *sensor_name*, computed as the (time total)/(number of times the phase completed).
sensor_name.active	Specifies the number of threads in the phase *sensor_name*, at the time the DMS statistics are gathered (the value may change over time).
sensor_name.maxActive	Specifies the maximum number of concurrent threads in the phase *sensor_name*, since the process started.

DMS Event Sensors A DMS **Event Sensor** is a Sensor that counts system events. Use a DMS Event Sensor to track system events that have a short duration, or where the duration of the event is not of interest but the occurrence of the event is of interest.

Table 9–2 describes the metric that is associated with an Event Sensor.

Table 9–2 DMS Event Sensor Metrics

Metric	Description
sensor_name.count	Specifies the number of times the event has occurred since the process started, where *sensor_name* is the name of the Event Sensor as specified in the DMS instrumentation API. Default: count is the default metric for an Event Sensor. No other metrics are available for an Event Sensor.

DMS State Sensors A DMS **State Sensor** is a Sensor to which you assign a precomputed value. The value can be of a variety of types, including: `int, long`, `double`, or `Object`. Use a State Sensor when you want to track system status information, or when you need a performance metric that is not associated with an event. For example, use State Sensors to represent queue lengths, pool sizes, buffer sizes, or host names.

Table 9–3 describes the State Sensor metrics. The `minValue` and `maxValue` optional metrics may only make sense when numeric values are associated with the State Sensor.

Table 9–3 DMS State Sensor Metrics

Metric	Description
sensor_name.value	Specifies the metric value for *sensor_name*, using the type assigned when *sensor_name* is created. Default: `value` is the default State metric.
sensor_name.count	Specifies the number of times *sensor_name* is updated.
sensor_name.minValue	Specifies the minimum value for *sensor_name* since startup.
sensor_name.maxValue	Specifies the maximum value this *sensor_name* since startup.

DMS Nouns

DMS Nouns (Nouns) organize performance data. Each Sensor, with its associated metrics is organized in a hierarchy according to Nouns. Nouns allow you to organize DMS metrics in a manner comparable to a directory structure in a file system. For example, Nouns can represent classes, methods, objects, queues, connections, applications, databases, or other objects that you want to measure.

A Noun `type` is a name that reflects the set of metrics being collected. For example, in the built-in metrics the Noun type `oc4j_servlet` represents the metrics collected for each servlet in each Web module within each J2EE application. And the Noun type `JVM` represents the set of metrics for each Java process (OC4J) currently running in the site.

> **Note:** In Appendix A, "Performance Metrics" the Noun type is called the metric table name.

The Noun naming scheme uses a '/' as the root of the hierarchy, with each Noun acting as a container under the root, or under its parent Noun.

See Also: Appendix A, "Performance Metrics"

DMS Object Relationships

This section describes the object relationships and attributes for DMS metrics, Sensors, and Nouns.

Table 9–4 describes the relationships between DMS objects. Figure 9–1 illustrates the relationships shown in Table 9–4 using a sample set of metrics.

Table 9–4 DMS Object Relationships and Attributes

Object	Contains	Attributes
Noun	Sensors or other Nouns	Name, Noun Type, Parent
Sensor	Metrics	Name, Description, Sensor Type There are three Sensor Types: PhaseEvent, Event, and State.
Metric	Value	Name, Units designation

Isolating Expensive Intervals Using PhaseEvent Metrics

Carefully consider the requirements for new metrics when you add DMS instrumentation. It is important to add a sufficient number of metrics to validate that your code is behaving as desired.

As a guide, try to observe the following rules when you add DMS metrics:

1. Add metrics only to provide an overview of the time the system spends in your block of code or module. You do not need to collect performance data for every method call, or for every distinct phase of your code or module.
2. When your code calls external code that you do not control, and that you expect could take a significant amount of time, add a PhaseEvent Sensor to track the start and the completion of the external code.

Following these rules provides the following benefits:

- Helps to limit the amount of information that DMS collects.
- Allows those analyzing the system to prove that a module gives the expected runtime performance.

- Ensures that people viewing DMS metrics can validate runtime performance without seeing an overwhelming amount of data.
- Allows those analyzing system performance to separate and track your module from other system modules that are either expensive or failure prone.

Adding DMS Instrumentation To Java Applications

You can collect performance information in Java applications by adding DMS instrumentation to existing applications or by creating new applications that include DMS instrumentation.

The DMS samples shown in this chapter are supplied on the Oracle Technology Network Web site,

`http://otn.oracle.com/tech/java/oc4j/demos/index.html`

The DMS `demo.zip` file includes a ready to deploy .ear file and source code with build instructions. The demo includes two servlets, `BasicBinomial.java` and `ImprovedBinomial.java`.

The BasicBinomial servlet shows how to use the DMS API to add DMS Sensors.

The ImprovedBinomial servlet expands on the BasicBinomial and illustrates measuring the improved code, as compared with the BasicBinomial. ImprovedBinomial servlet also shows how to add more costly metrics that gather more detailed information. You can use ImprovedBinomial to compare the cost of enabling heavy instrumentation and the use of the DMS features that conditionally use instrumentation with the DMS SensorWeight feature.

Refer to the sample code for full details on the examples in this chapter.

To use DMS instrumentation, add DMS calls by performing the following steps:

- Including DMS Imports
- Organizing Performance Data
- Defining and Using Metrics for Timing
- Defining and Using Metrics for Counting
- Defining and Using Metrics for Recording Status Information

Including DMS Imports

To use DMS you need to add DMS imports. The following example shows the imports that the sample application `BasicBinomial.java` requires.

```
import oracle.dms.instrument.DMSConsole;
import oracle.dms.instrument.Event;
import oracle.dms.instrument.Noun;
import oracle.dms.instrument.PhaseEvent;
import oracle.dms.instrument.State;
import oracle.dms.instrument.Sensor;
```

Organizing Performance Data

Define DMS Nouns to organize Sensors and their associated metrics. DMS Nouns organize Sensors in a tree hierarchy in a manner comparable to a directory structure in a file system, starting with a root at the top of the tree.

Example 9–2 shows a section of code using `Noun.create()` from the `BasicBinomial.java`.

In Example 9–2, `MathSeries` specifies the Noun type. The Noun type is a name that reflects the set of metrics being collected. For example, `MathSeries` represents the metrics collected for the sample application containing a Binomial series computation. `AggreSpy` displays Sensors using the same Noun type together.

It is good practice to only use Noun types for Nouns that directly contain Sensors. When a Noun contains only Nouns, as in the Noun `dmsDemo`, and does not directly contain Sensors, `AggreSpy` displays the Noun type as a metric table, with no metrics. Example 9–2 shows the `dmsDemo` Noun that includes a Noun, `BasicBinomial`, but no Sensors. When the Noun type is not included for such a Noun, `AggreSpy` does not display a metric table associated with the Noun (the Noun's children, with their associated Sensors are shown).

> **Note:** Start Noun type names with a capital letter to distinguish them from other DMS names.

Example 9–2 Using Noun.create To Organize Sensors

```
private Noun binRoot;    // root of the Binomial series DMS metrics
..
.
binRoot = Noun.create("/BasicBinomial", "MathSeries");
```

See Also: "DMS Naming Conventions" on page 9-17

Defining and Using Metrics for Timing

To create metrics that time an interval, define and use a PhaseEvent Sensor as follows:

- Defining PhaseEvent Sensors
- Using PhaseEvent Sensors

Defining PhaseEvent Sensors

Example 9–3 shows the DMS calls that declare and create the `computeSeries` PhaseEvent Sensor. This code defines a DMS metric named `/Binomial/computeSeries.time`.

PhaseEvent Sensors support a set of optional metrics, along with the default metric `computeSeries.time` (representing the time, as measured between the `PhaseEvent start()` and `stop()` calls). Associate optional metrics with PhaseEvent Sensors individually or as a complete set. Table 9–1 shows the available metrics for a PhaseEvent Sensor. The `binComp.deriveMetric(Sensor.all)` call in Example 9–3 enables all the supported optional metrics.

Note: Using the method `deriveMetric(Sensor.all)` is recommended for adding optional metrics. Using this method with `Sensor.all` adds all metrics; this is good practice since the list of optional metrics could change in a future Oracle Application Server release.

Example 9–3 Defining PhaseEvent Sensors

```
private PhaseEvent binComp; // Time to compute Binomial series.
.
.
.
binComp = PhaseEvent.create(binRoot, "computeSeries",
                                  "Time to compute a Binomial series");
binComp.deriveMetric(Sensor.all);
```

Using PhaseEvent Sensors

To use a PhaseEvent Sensor, an application calls the `start()` method to indicate the beginning of a phase, and subsequently calls the `stop()` method to indicate the completion of the phase.

Example 9–4 shows a code segment from `BasicBinomial.java` that uses the `start()` and `stop()` methods for the `/Binomial/computeSeries.time` metric. The `long` value named `token` that is returned from the PhaseEvent `start()` method must be passed to the corresponding PhaseEvent `stop()` method. This value is a timestamp representing the start time. Passing this value to the `stop()` method allows DMS to compute the PhaseEvent duration.

Note: To assure that PhaseEvents are stopped, each PhaseEvent `start()` method, together with the code to be measured should be in a `try` block with the PhaseEvent `stop()` method in a corresponding `finally` block, as shown in Example 9–4.

Example 9–4 Using start() and stop() With PhaseEvent Sensors

```
long token = 0; // DMS
try {
    token = binComp.start(); // DMS
    BigInteger bins[] = bin(length);
    out.println("<H2>Binomial series for " +  length + "</H2>");
    for (int i = 0; i < length; i++)
        out.println("<br>" + bins[i]);
    }
    finally {
        binComp.stop(token); // DMS
        out.close();
    }
```

Defining and Using Metrics for Counting

To create metrics that count the occurrences of an event, define and use an Event Sensor as follows:

- Defining Event Sensors
- Using Event Sensors

Defining Event Sensors

Example 9–5 shows the DMS calls that define an Event Sensor. This code allocates a counter and defines a DMS metric named `/Binomial/loops.count`.

Example 9–5 Defining Event Sensors

```
private Event binLoop;   // Loops needed for Binomial series.
.
.
.

binLoop = Event.create(binRoot, "loops", "Iterations to compute series");
```

Using Event Sensors

DMS increments a counter when an application calls the `occurred()` method for an Event Sensor. Example 9–6 shows the `occurred()` call for an Event Sensor that increments the `/Binomial/loops.count` metric.

Example 9–6 Using occurred() With Event Sensors

```
binLoop.occurred();
```

Defining and Using Metrics for Recording Status Information

DMS captures status information with State Sensors. State sensors are general sensors, in that you use them to track Java Objects. DMS stores a reference to the Java Object and calls `toString()` on the object when the DMS value is sampled.

To create metrics that record status information define and use a State Sensor as follows:

- Defining State Sensors
- Using State Sensors

Defining State Sensors

State Sensors support a set of optional DMS metrics, along with the default metric `value`. You associate optional metrics with State Sensors individually or as a complete set. Table 9–3 shows the available metrics for a State Sensor. Example 9–3 shows how to enable the optional metrics.

Example 9–7 shows the DMS calls that declare and create a State Sensor. This code defines a DMS metric named `/Binomial/lastComputed.value`. When you define a State Sensor, using the empty string in the `create()` method indicates that no units are associated with the State Sensor, as shown in Example 9–7.

Example 9–7 Defining State Sensors

```
private State binLast;   // Value of the last computed element in series.
.
.
.
binLast = State.create(binRoot, "lastComputed", "",
                       "Value of last computed series element", "0");
```

Using State Sensors

When an application calls a State Sensor's `update()` method, DMS updates the value of the state sensor. Example 9–8 shows the `update()` call for a State Sensor that updates the `/Binomial/lastComputed.value` metric.

Example 9–8 Using update() With State Sensors

```
binLast.update(bins[k-1].toString());
```

Validating and Testing Applications Using DMS Metrics

You should test and verify the accuracy of the metrics that you add to Java applications.

This section covers the following topics:

- Validating DMS Metrics
- Testing DMS Metrics For Efficiency

Validating DMS Metrics

Use the `dmstool` and the other available DMS monitoring tools to verify and test new metrics.

Try to validate the following for new metrics:

- Do expected metrics appear in the display? Test this by examining the code to make sure that all the metric names added using DMS instrumentation appear in your display or saved set of metrics.
- Do unexpected metrics appear in the display? Verify that you have only added the metrics that you planned to add.
- Are the metric values you see within reasonable ranges? Usually, upper and lower bounds for metrics can be established. You then test that the reported values for metrics do not exceed the expected bounds.

  For example, a "size of pool" metric should never report a negative value.
- Are metric values accurate? This can be difficult to test; however, if an alternate means of measuring a particular metric is available then use it to verify metric values. For example, if you submit a known number of requests to a server and measure total time for the experiment, then you predict correct values for the relevant metrics and compare them with the actual monitored values. As another example, you can verify an Event Sensor `count` metric by examining records that you write to a log file or to the console.

  Check for timing inaccuracies that may apply for the metrics. Timing inaccuracies may be caused when low-resolution clocks time metrics for an interval of short duration. For example on Windows systems, the default Java clock advances only once every 15 milliseconds. DMS metrics reported for brief events on these systems must be analyzed with care.
- When integrating DMS instrumentation with an existing package or when implementing a new feature, you should consider insulating a previously working system. For example, you could include an option to enable and disable new DMS metrics.

Testing DMS Metrics For Efficiency

The use of DMS metrics has some influence on application performance. When adding metrics, note the following:

- The processing required for computing and storing metrics can slow down the execution of an application. DMS is fast and efficient, but it does have some required overhead cost. In addition, DMS cannot prevent developers from using the DMS API inefficiently. Therefore, before adding DMS instrumentation, establish reasonable expectations. After completing the implementation, measure the actual costs and compare them to your expectations. Be prepared to make changes to the implementation to reduce overhead costs until the measurements agree with expectations.
- Make sure that new metrics are accurate. For most applications using DMS metrics, accuracy is more important than the performance cost of adding the DMS instrumentation. New DMS metrics should provide reliable and useful information.
- DMS provides the `DMSConsole.getSensorWeight()` method to help you control the use of metrics. The central setting is an advisory measurement level that DMS does not enforce. To control which metrics to include, at runtime, the code must test the value for `SensorWeight` to determine whether to make DMS calls.
- When integrating DMS instrumentation with an existing package or when implementing a new feature, you should consider insulating a previously working system. For example, you could include an option to enable and disable new DMS metrics.
- Worrying about performance too soon often leads to costly design and implementation errors. According to Donald Knuth, "Premature optimization is the root of all evil".
- You should run your performance tests with and without DMS enabled. If your tests show unacceptable results with DMS enabled, then you may want to re-design or re-implement metrics.

Understanding DMS Security Considerations

DMS metrics do not support user based access to DMS reports. When you define and use a DMS metric, the metric is available to any administrator that has access to DMS metrics. This means when you add DMS metrics, it is good practice to avoid placing customer sensitive information in the metrics.

When you add DMS instrumentation, the following users have access to the DMS metrics that you create:

- Applications running in the same OC4J instance can access the DMS metrics.
- All users that have access to the `dmstool` command, or the `Spy` or `AggreSpy` Servlets have access to the metrics (by default this is limited to Administrators).

See Also:

- "AggreSpy URL and Access Control" on page 2-8
- "Access Control for dmstool" on page 2-10

Advanced DMS Topics

This section covers advanced DMS topics, including the following:

- DMS Naming Conventions
- DMS Coding Recommendations
- Resetting and Destroying Sensors
- Dumping DMS Metrics To Files
- Conditional Instrumentation Using DMS Sensor Weight

DMS Naming Conventions

Certain guidelines apply for defining DMS names. By following these guidelines, people viewing DMS metric reports can easily understand metrics across applications and across Oracle Application Server components.

Note: View the naming conventions as guidelines; for each rule there may be an exception. In applying the rules, try to be as clear as possible, if there is a conflict, you may need to make an exception.

This section covers the following topics:

- General DMS Naming
- General DMS Naming Rules and Character Sets
- Noun and Noun Type Naming Rules
- Sensor Naming Rules

General DMS Naming

DMS metric names consist of a Sensor name, plus the "." character, plus the default metric or optional derivation. For example, the names: `computeSeries.time`, `loops.count`, and `lastComputed.value` are valid DMS metric names.

A Sensor base name is a simple string, not including the "." or the derivation. For example `computeSeries`, `loops`, and `lastComputed` are Sensor base names. A Sensor full name consists of the Sensor base name, preceded by the full name of its associated Noun, and a delimiter. For example, `/Binomial/computeSeries`, `/Binomial/loops`, and `/Binomial/lastComputed`.

A Noun base name is a simple string, not including a delimiter. For example `Binomial` is a noun base name. A Noun full name consists of the Noun base name, preceded by the full name of its parent, and a delimiter. For example `/Binomial` is a full Noun name.

General DMS Naming Rules and Character Sets

DMS names should be as compact as possible. Whenever possible, when you define Noun and Sensor names, avoid special characters such as white space, slashes, periods, parenthesis, commas, and control characters.

DMS replaces special characters with the "_" underscore character.

Note: Oracle Application Server includes a number of built-in metrics. The Oracle Application Server built-in metrics do not always follow the DMS naming rules.

Noun and Noun Type Naming Rules

A Noun name should be a name which identifies a specific entity of interest.

Noun types should have names which clearly reflect the set of metrics being collected. For example, Servlet is the type for a Noun under which the metrics that are specific to a given servlet fall.

Noun type names should start with a capitol letter to distinguish them from other DMS names. All Nouns of a given type should contain the same set of sensors.

Sensor Naming Rules

1. Sensor names should be descriptive, but not redundant. Sensor names should not contain any part of the Noun name hierarchy, or type, as this is redundant. Sensor names should avoid containing the specification of the units for the individual metrics. Where multiple words are required to describe a Sensor, the first word should start with a small letter, and the following words should start with capitol letters. For example `computeSeries`.
2. Event Sensor and PhaseEvent Sensor names should have the form *verbNoun* where *verb* and *Noun* are interpreted as defined by English grammar. For example, `activateInstance` and `runMethod`. When a PhaseEvent monitors a function, method, or code block, it should be named to reflect the task performed as clearly as possible.
3. The name of a State Sensor should be a Noun, possibly preceded by an adjective, which describes the semantics of the value which is tracked with this State. For example, `lastComputed, totalMemory`, `port`, `availableThreads, activeInstances`.

DMS Coding Recommendations

The following list includes coding recommendations for working with DMS.

1. There is a global name space for DMS metrics. When you create a new Noun Sensor (PhaseEvent, Event, or State), its full name must not conflict with names in use by Oracle built-in metrics, or by other applications.

   See Also: "General DMS Naming" on page 9-17

2. Be sure all PhaseEvents are stopped. If the code block to be measured is not in a `try` block, then put it in a `try` block that includes PhaseEvent's `start()`. Put the PhaseEvent's `stop()` in a `finally` block.

   See Also: "Using PhaseEvent Sensors" on page 9-11

3. Use the DMS naming conventions.

   See Also: "DMS Naming Conventions" on page 9-17

4. Avoid creating any DMS Sensor or Noun more than once. The DMS API allows this, and avoids creation of multiple objects, but DMS performs lookups for each subsequent call. Thus, you should define Sensors and Nouns during static initialization, or in the case of a Servlet, in the `init()` method.
5. Assign a type for each Noun. If no type is assigned, the type is given the value "n/a" (not available). Nouns with the type specified as "n/a" are not shown in the `Spy` or `AggreSpy` display.
6. Avoid creating PhaseEvents which do not measure a section of code that is expensive under some set of conditions.
7. The DMS API calls are threadsafe; they provide sufficient synchronization to prevent races and access bugs.

Resetting and Destroying Sensors

The Sensor abstract class provides methods that you use to control PhaseEvent, Event, and State Sensors. The `reset()` method resets a Sensor's metrics to initial values. The `destroy()` method removes a Sensor from DMS and releases references to its underlying resources.

Dumping DMS Metrics To Files

In a Java application, use the following methods to dump DMS metrics to a file.

The following code appends the current metrics to `./dms.log`:

```
DMSConsole cons = new DMSConsole();
cons.dump();
```

The following code allows you to append or replace the contents of the specified file with the current metrics:

```
DMSConsole cons2 = new DMSConsole();
DMSConsole.dump("dmsmathseries.log", true, true);
```

The first argument specifies the file path name, the second argument specifies the output format, and the third argument specifies if the output is appended to the file or replaces the contents of the file.

Conditional Instrumentation Using DMS Sensor Weight

Use the DMS sensor weight feature to conditionally apply metrics. With sensor weight, you specify that applications execute expensive instrumentation only when the sensor weight is set to a particular value. Using this feature, enables you to include expensive metrics that you may only need for debugging.

Example 9–9 shows how to use `DMSConsole.getSensorWeight()` to test the value of the sensor weight, and optionally define and use a metric.

The sensor weight is set globally using the `oracle.dms.sensors` property on the command-line. Set this property using the OC4J startup options. Supported values for this property include: `none`, `normal`, `heavy`, and `all`.

Example 9–9 Using SensorWeight for Conditional Instrumentation

```
 /* DMS Method
   *
   * If the SensorWeight is high enough, return a phase with the
   * parameter in the name. Otherwise, return null.
   */
PhaseEvent heavyPhase(String param) {
PhaseEvent pe = null;
if (DMSConsole.getSensorWeight() > DMSConsole.NORMAL) {
   Noun base = Noun.create(binRoot, param, "MathSeries");
   pe = PhaseEvent.create(base, "computeSeries",
                                   "Time to compute a Binomial series");
   pe.deriveMetric(Sensor.all);
   }
return pe;
```

See Also: "Setting Java Command Line Options (Using JVM and OC4J Performance Options)" on page 6-3

10

Database Tuning Considerations

To achieve optimal performance in Oracle Application Server, for applications that use the database, the database tables you access need to be designed with performance in mind, and you need to monitor and tune the database server to assure that the system is performant. This chapter describes some of the `init.ora` parameters that may need to be tuned in a backend Oracle Database Server.

This chapter covers the following:

- Tuning init.ora Database Parameters
- Tuning Redo Logs Location and Sizing

See Also: *Oracle9i Database Performance Tuning Guide and Reference*

Tuning init.ora Database Parameters

Table 10–1 shows tuning information for several the `init.ora` database initialization parameters.

Table 10–1 Important init.ora Tuning Parameters

init.ora Parameter	Description
DB_BLOCK_SIZE	Sets the database block size. OLTP applications usually benefit from smaller block sizes, DSS applications usually benefit from larger block sizes. This parameter can only be set when the database is created, and defaults to the minimum value of 2K. See Also: table 15-4, "Block Size Advantages and Disadvantages" in the *Oracle9i Database Performance Tuning Guide and Reference*
DB_CACHE_SIZE	Sets the size of the buffer cache in the SGA. Larger cache sizes generally reduce the number of disk reads and writes. However, a large cache may take up too much memory and induce memory paging or swapping. On most operating systems, the disadvantage of paging significantly outweighs the advantage of a large SGA. The V$DB_CACHE_ADVICE view provides predictions of behavior using various buffer cache sizes. Note that the initialization parameter DB_CACHE_ADVICE must be set to ON before V$DB_CACHE_ADVICE will gather statistics. The parameter DB_BLOCK_BUFFERS has been deprecated. See Also: "Memory Configuration and Use" in the *Oracle9i Database Performance Tuning Guide and Reference*.
JAVA_POOL_SIZE	If you are using Java stored procedures, then this parameter should be set depending on the actual requirements of memory for the Java environment. The V$SGASTAT view will show the current java pool allocation.
PGA_AGGREGATE_TARGET	Specifies the target aggregate PGA memory available to all server processes attached to the instance. See Also: the chapter, "Memory Configuration and Use" in the *Oracle9i Database Performance Tuning Guide and Reference* for information on PGA memory management.
PROCESSES	Sets the maximum number of operating system processes that can be connected to Oracle concurrently. The value of this parameter must be 6 or greater (5 for the background processes plus 1 for each user process). For example, if you plan to have 50 concurrent users, set this parameter to at least 55. Many other initialization parameter values are deduced from this value.

Table 10–1 (Cont.) Important init.ora Tuning Parameters

init.ora Parameter	Description
SHARED_POOL_SIZE	Sets the size of the shared pool in the SGA. The main components of the shared pool are the library cache and the dictionary cache. A cache miss on the data dictionary cache or library cache is more expensive than a miss on the buffer cache. For this reason, the shared pool should be sized to ensure that frequently used library and dictionary data is cached. The statistic that shows the amount of reloading (that is, reparsing) of a previously cached SQL statement that was aged out of the cache is the RELOADS column in the V$LIBRARYCACHE view. In an application that reuses SQL effectively, on a system with an optimal shared pool size, the RELOADS statistic will have a value near zero. Another key statistic is the amount of free memory in the shared pool at peak times. The amount of free memory can be queried from V$SGASTAT, looking at the free memory for the shared pool. Optimally, free memory should be as low as possible, without causing any reloads on the system. See Also: the chapter, "Memory Configuration and Use" in the *Oracle9i Database Performance Tuning Guide and Reference*.
UNDO_TABLESPACE, UNDO_MANAGEMENT	Undo space can be managed with either rollback segments or undo tablespaces. Good performance can be achieved by either method, however, the use of rollback segments for managing undo space will be deprecated in a future release. Oracle strongly recommends that you use automatic undo management (UNDO_MANAGEMENT = AUTO) and manage undo space using an UNDO_TABLESPACE. For backward compatibility reasons, the default value of UNDO_MANAGEMENT is MANUAL. See Also: *Oracle9i Database Administrator's Guide* for additional information on undo space management.

Tuning Redo Logs Location and Sizing

Managing the database I/O load balancing is a non-trivial task. However, tuning the redo log options can provide performance improvement for applications running in an Oracle Application Server environment, and in some cases, you can significantly improve I/O throughput by moving the redo logs to a separate disk.

The size of the redo log files can also influence performance, because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance. Small log files can increase checkpoint activity and reduce performance. Because the recommendation on I/O distribution for high performance is to use separate disks for the redo log files, there is no reason not to make them large. A potential problem with large redo log files is that these are a single point of failure if redo log mirroring is not in effect.

It is not possible to provide a specific size recommendation for redo log files, but redo log files in the range of a hundred megabytes to a few gigabytes are considered reasonable. Size your online redo log files according to the amount of redo your system generates. A rough guide is to switch logs at most once every twenty minutes. Set the initialization parameter `LOG_CHECKPOINTS_TO_ALERT = true` to have checkpoint times written to the alert file.

The complete set of required redo log files can be created during database creation. After they are created, the size of a redo log size cannot be changed. However, new, larger files can be added later, and the original (smaller) ones can subsequently be dropped.

> **See Also:** The chapters, "Building a Database for Performance" and "I/O Configuration and Design" in the *Oracle9i Database Performance Tuning Guide and Reference*

A

Performance Metrics

This appendix lists built-in metrics that can help you analyze Oracle Application Server performance. The metrics fall into several distinct areas, such as Oracle HTTP Server, Oracle Application Server Containers for J2EE (OC4J), and Portal. Each table in this chapter lists the metrics that are included in a corresponding Dynamic Monitoring Services metric table.

This appendix contains:

- Oracle HTTP Server Metrics
- JVM Metrics
- JDBC Metrics
- OC4J Metrics
- OC4J JMS Metrics
- OC4J Task Manager Metrics
- mod_plsql Metrics
- Portal Metrics
- JServ Metrics
- Oracle Process Manager and Notification Server Metrics
- Oracle Application Server Discoverer Metrics

Oracle HTTP Server Metrics

The tables, Table A–1, Table A–4, Table A–5 describe the Oracle HTTP Server metrics.

The metric table name is `ohs_server`.

Table A–1 HTTP Server Metrics (ohs_server)

Metric	Description	Unit
`connection.active`	Number of connections currently open	threads
`connection.avg`	Average time spent servicing HTTP connections	usecs
`connection.maxTime`	Maximum time spent servicing any HTTP connection	usecs
`connection.minTime`	Minimum time spent servicing any HTTP connection	usecs
`connection.time`	Total time spent servicing HTTP connections	usecs
`handle.active`	Child servers currently in the handle processing phase	threads
`handle.avg`	Average time spent in module handler	usecs
`handle.completed`	Number of times the handle processing phase has completed	ops
`handle.maxTime`	Maximum time spent in module handler	usecs
`handle.minTime`	Minimum time spent in module handler	usecs
`handle.time`	Total time spent in module handler	usecs
`request.active`	Child servers currently in the request processing phase	threads
`request.avg`	Average time required to service an HTTP request	usecs
`request.completed`	Number of HTTP request completed	ops
`request.maxTime`	Maximum time required to service an HTTP request	usecs
`request.minTime`	Minimum time required to service an HTTP request	usecs
`request.time`	Total time required to service HTTP requests	usecs

Oracle HTTP Server Child Server Metrics

Table A–2 describes the child server metrics.

The metric table name is `ohs_child`.

Table A–2 Oracle HTTP Server Child Server Metrics (ohs_child)

Metric	Description	Unit
pid.value	Process ID	
slot.value	Slot	
status.value		
time.value		
url.value		

Oracle HTTP Server Responses Metrics

The Oracle HTTP Server responses metrics are included in the metric table named `ohs_responses`. This metric table includes one metric containing the count, number of times the response was generated, for each HTTP response type.

For example, `Success_OK_200.count: 28 ops`.

Oracle HTTP Server Virtual Host Metrics

The Oracle HTTP Server `ohs_vhostSet` and `ohs_virtualHost` metric tables contain information on virtual host names and locations, and request and response metrics.

Table A–3 Oracle HTTP Server Virtual Host Metrics (ohs_virtualHost)

Metric	Description	Unit
`request.active`	Active requests	threads
`request.avg`	Average time for request processing	usecs
`request.completed`	Number of completed requests	ops
`request.maxTime`	Maximum time to complete a request	usecs
`request.minTime`	Minimum time to complete a request	usecs
`request.time`		usecs
`responseSize.value`		bytes
`vhostType.value`		

Aggregate Module Metrics

Table A–4 HTTP Server Apache/Modules Metrics

Metric	Description	Unit
numMods.value	Number of loaded modules	

HTTP Server Module Metrics

There is one set of metrics for each module loaded into the server.

The metric table name is `ohs_module`.

Table A–5 HTTP Server Apache/Modules/mod_*.c Metrics (ohs_module)

Metric	Description	Unit
decline.count	Number of requests declined	ops
handle.active	Number of requests currently being handled by this module	requests
handle.avg	Average time required for this module	usecs
handle.completed	Number of requests handled by this module	ops
handle.maxTime	Maximum time required for this module	usecs
handle.minTime	Minimum time required for this module	usecs
handle.time	Total time required for this module	usecs

Oracle HTTP Server mod_oc4j Metrics

Table A–6 shows the mod_oc4j Failure Causes metrics. This table represents the categorization of errors that return an `INTERNAL_SERVER_ERROR` to the client.

The metric table name is `mod_oc4j_request_failure_causes`.

Table A–6 HTTP Server mod_oc4j Request Failure Causes Metrics

Metric	Description	Unit
`IncorrectReqInit.count`	The total number of times an internal error occurred. There could be a number of reasons, including: mod_oc4j not finding a connection endpoint, configuration errors, and others.	ops
`Oc4jUnavailable.count`	The total number of times that an oc4j JVM could not be found to service requests.	ops
`UnableToHandleReq.count`	The total number of times mod_oc4j declined to handle a request.	ops

Table A–7 shows the mod_oc4j Mount Point metrics. There is one mount point metric table for each mount point specified in mod_oc4j.conf. This table includes a set of metrics for each mount point specified, with each set grouped under the mntPt*id*. Where *id* is an integer that is automatically generated during module initialization.

The metric table name is `mod_oc4J_mount_pt_metrics`.

Table A–7 HTTP Server mod_oc4j Mount Point Metrics

Metric	Description	Unit
`Destination.value`	Specifies the destination name. For example, with: `Oc4jMount /j2ee/* home` The `Destination.value` would be `home`	String
`ErrReq.count`	Specifies the total number of requests, both session and non-session, that mod_oc4j failed to route to an OC4J.	ops
`ErrReqNonSess.count`	Specifies the total number of non session requests that mod_oc4j failed to route to an oc4j process.	ops
`ErrReqSess.count`	Specifies the total number of session requests that mod_oc4j failed to route to an OC4J process.	ops
`Failover.count`	Specifies the total number of failovers for both nonsession and session requests.	ops
`Name.value`	Specifies the echo of the value specified as the path for Oc4jMount directive in mod_oc4j.conf. DMS changes certain characters, including: '/' and '*' to '_' . To preserve the actual path names specified, an internal table containing a mapping between mntPt*id* and the actual path name is created during `mod_oc4j` initialization. For example, with: `Oc4jMount /j2ee/* home` `Name.value` would be `/j2ee/*`	String
`NonSessFailover.count`	Specifies the total number of failovers for nonsession requests. For example, Assume that this mount point was serviced by an OC4J Island with three JVM's (JVM1, JVM2 and JVM3). A new non session request is routed to JVM1. JVM1 fails to service the request, and the request is failed over to JVM2. JVM2 fails to service the request, and so the request is failed over to JVM3. At this point the `NonSessFailover.count` is incremented by 2.	ops

Table A–7 (Cont.) HTTP Server mod_oc4j Mount Point Metrics

Metric	Description	Unit
SessFailover.count	Specifies the total number of failovers for session requests. For example, Let us assume that this mount point was serviced by an OC4J Island with three JVM's (JVM1, JVM2 and JVM3). A session request is routed to JVM1. JVM1 fails to service the request. So, the request is failed over to JVM2. At this point the SessFailover.count is incremented by 1. JVM2 fails to service the request, and so the request is failed over to JVM3. At this point the SessFailover.count is incremented by 2.	ops
SucReq.count	Specifies the total number of requests, both session and non-session, that mod_oc4j successfully routed to an OC4J instance.	ops
SucReqNonSess.count	Specifies the total number of non session requests that mod_oc4j successfully routed to an OC4J process.	ops
SucReqSess.count	Specifies the total number of session requests that mod_oc4j successfully routed to an OC4J process.	ops

Table A–8 shows the mod_oc4j Destination Metrics. This table includes a set of metrics for a specific destination. Each destination can have multiple mount points. There is one mntPts subtree for each mount point specified in mod_oc4j.conf.

The metric table name is `mod_oc4J_destination_metrics`.

Table A–8 HTTP Server mod_oc4j Destination Metrics

Metric	Description	Unit
ErrReq.count	Specifies the total number of requests, both session and non-session, that mod_oc4j failed to route to an OC4J.	ops
ErrReqNonSess.count	Specifies the total number of non session requests that mod_oc4j failed to route to an OC4J process.	ops
ErrReqSess.count	Specifies the total number of session requests that mod_oc4j failed to route to an OC4J process.	ops
Failover.count	Specifies the total number of failovers for both nonsession and session requests.	ops
JVMCnt.value	Specifies the total number of routable OC4J JVMs that belong to this destination.	Number of JVMs
Name.value	Specifies the echo of the value specified as destination for Oc4jMount directive in mod_oc4j.conf, a single destination may appear several times in mod_oc4j.conf. Example: Oc4jMount /j2ee/* home,oc4jinstance2 Name.value would be home,oc4jinstance2	String

Table A–8 (Cont.) HTTP Server mod_oc4j Destination Metrics

Metric	Description	Unit
`NonSessFailover.count`	Specifies the total number of failovers for non session requests.	ops
`SessFailover.count`	Specifies the total number of failovers.	ops
`SucReq.count`	Specifies the total number of requests, both session and non-session, that mod_oc4j successfully routed to an OC4J.	ops
`SucReqNonSess.count`	Specifies the total number of non session requests that mod_oc4j successfully routed to an OC4J process.	ops
`SucReqSess.count`	Specifies the total number of session requests that mod_oc4j successfully routed to an OC4J process.	ops

JVM Metrics

There is one set of metrics for each Java process (OC4J or JServ) currently running in the site. The metric table name is `JVM`.

Table A–9 JVM Metrics (JVM)

Metric	Description	Unit
`activeThreadGroups.value`	The number of active thread groups in the JVM	integer
`activeThreadGroups.minValue`	The minimum number of active thread groups in the JVM	integer
`activeThreadGroups.maxValue`	The maximum number of active thread groups in the JVM	integer
`activeThreads.value`	The number of active threads in the JVM	threads
`activeThreads.minValue`	The minimum number of active threads in the JVM	threads
`activeThreads.maxValue`	The maximum number of active threads in the JVM	threads
`upTime.value`	Up time for the JVM	msecs
`freeMemory.value`	The amount of heap space free in the JVM	KB
`freeMemory.minValue`	The minimum amount of heap space free in the JVM	KB
`freeMemory.maxValue`	The maximum amount of heap space free in the JVM	KB

Table A–9 (Cont.) JVM Metrics (JVM)

Metric	Description	Unit
totalMemory.value	The total amount of heap space in the JVM	KB
totalMemory.minValue	The minimum amount of total heap space in the JVM	KB
totalMemory.maxValue	The maximum amount of total heap space in the JVM	KB

JDBC Metrics

The following tables list the Oracle Application Server JDBC metrics.

JDBC Driver Metrics

There is one set of JDBC Driver metrics per JVM.

The metric table name is `JDBC_Driver`.

Table A–10 /JDBC/Driver - JDBC_Driver Metrics

Metric	Description	Unit
ConnectionCloseCount.count	Total number of connections that have been closed.	ops
ConnectionCreate.active	Current number of threads creating connections.	ops
ConnectionCreate.avg	Average time spent creating connections.	msecs
ConnectionCreate.completed	Number of times this PhaseEvent has started and ended.	ops
ConnectionCreate.maxTime	Maximum time spent creating connections.	msecs
ConnectionCreate.minTime	Minimum time spent creating connections.	msecs
ConnectionCreate.time	Time spent creating connections.	msecs
ConnectionOpenCount.count	Total number of connections that have been opened.	ops

JDBC Data Source Metrics

The metric table name is `JDBC_DataSource`.

There is one set of data source metrics per data source.

Table A–11 /JDBC/`data-source-name` - JDBC_Data Source Metrics

Metric	Description	Unit
CacheFreeSize.value	Number of free slots in the connection cache.	
CacheGetConnection.avg	Average time spent getting a connection from the cache.	msecs
CacheGetConnection.completed	Number of times this PhaseEvent has started and ended.	ops

Table A–11 (Cont.) /JDBC/*data-source-name* - JDBC_Data Source Metrics

Metric	Description	Unit
CacheGetConnection.maxTime	Maximum time spent getting a connection from the cache.	msecs
CacheGetConnection.minTime	Minimum time spent getting a connection from the cache.	msecs
CacheGetConnection.time	Time spent getting a connection from the cache or not.	msecs
CacheHit.count	Number of times a request for a connection has been satisfied from the cache.	
CacheMiss.count	Number of times a request for a connection failed to be satisfied from the cache.	
CacheSize.value	Total size of the connection cache.	

JDBC Driver Specific Connection Metrics

There is one set of JDBC Connection metrics per connection.

The metric table name is `JDBC_Connection`.

Table A–12 /JDBC/Driver/*CONNECTION* - JDBC Driver Connection Metrics

Metric	Description	Unit
CreateNewStatement.avg	Average time spent creating a new statement.	msecs
CreateNewStatement.completed	Number of times a request for a statement failed to be satisfied from the cache.	ops
CreateNewStatement.maxTime	Maximum time spent creating a new statement.	msecs
CreateNewStatement.minTime	Minimum time spent creating a new statement.	msecs
CreateNewStatement.time	Time spent creating a new statement (this does not include the time required to parse the statement. For information on the metric that includes the parse time see `Execute.Time` in Table A–14).	msecs
CreateStatement.avg	Average time spent getting a statement from the statement cache.	msecs
CreateStatement.completed	Number of times a request for a statement was satisfied from the cache.	ops
CreateStatement.maxTime	Maximum time spent getting a statement from the statement cache.	msecs
CreateStatement.minTime	Minimum time spent getting a statement from the statement cache.	msecs
CreateStatement.time	Time spent getting a statement from the statement cache.	msecs
LogicalConnection.value	If this is a physical connection, then this refers to its logical connection, if any.	

JDBC Data Source Specific Connection Metrics

There is one set of JDBC data source specific connection metrics per data source per connection. The metric table name is `JDBC_Connection`.

Table A–13 /JDBC/`data-source-name`/`CONNECTION` - JDBC Datasource Connection Metrics

Metric	Description	Unit
`CreateNewStatement.avg`	Average time spent creating a new statement.	msecs
`CreateNewStatement.completed`	Number of times a request for a statement failed to be satisfied from the cache.	ops
`CreateNewStatement.maxTime`	Maximum time spent creating a new statement.	msecs
`CreateNewStatement.minTime`	Minimum time spent creating a new statement.	msecs
`CreateNewStatement.time`	Time spent creating a new statement (this time does not include the time required to parse the statement. For information on the metric that includes the parse time see `Execute.Time` in Table A–15).	msecs
`CreateStatement.avg`	Average time spent getting a statement from the statement cache.	msecs
`CreateStatement.completed`	Number of times a request for a statement was satisfied from the cache.	ops
`CreateStatement.maxTime`	Maximum time spent getting a statement from the statement cache.	msecs
`CreateStatement.minTime`	Minimum time spent getting a statement from the statement cache.	msecs
`CreateStatement.time`	Time spent getting a statement from the statement cache.	msecs
`LogicalConnection.value`	If this is a physical connection, then this refers to its logical connection, if any.	

JDBC Driver Statement Metrics

There is a set of statement metrics per connection per statement. The metric table name is `JDBC_Statement`.

Note: The JDBC statement metrics are only available for JDBC connections that have enabled statement caching, and set the property `oracle.jdbc.DMSStatementCachingMetrics` to the value `true`. When JDBC statement caching is disabled, you can make the JDBC statement metrics available by setting the property `oracle.jdbc.DMSStatementMetrics` to `true`. To improve performance and to avoid collecting expensive metrics, by default these properties are both set to `false`.

Table A–14 /JDBC/Driver/`CONNECTION`/`STATEMENT` JDBC Statement Metrics

Metric	Description	Unit
`Execute.time`	The time this statement has spent executing the SQL including the first fetch and the time required to parse the statement.	msecs
`Fetch.time`	The time this statement has spent in other fetches.	msecs
`SQLText.value`	The SQL being executed.	

See Also: "Setting the OC4J JDBC DMS Statement Metrics Option" on page 6-9

JDBC Data Source Statement Metrics

The metric table name is `JDBC_Statement`. There is a set of statement metrics per data source per connection per statement.

Note: The JDBC statement metrics are only available for JDBC connections that have enabled statement caching and set the property `oracle.jdbc.DMSStatementCachingMetrics` to the value `true`. When JDBC statement caching is disabled, you can make the JDBC statement metrics available by setting the property `oracle.jdbc.DMSStatementMetrics` to `true`. To improve performance and to avoid collecting expensive metrics, by default these properties are set to `false`.

Table A–15 /JDBC/`data-source-name`/`CONNECTION`/`STATEMENT` JDBC Statement Metrics

Metric	Description	Unit
`Execute.time`	The time this statement has spent executing the SQL including the first fetch and the time required to parse the statement.	msecs
`Fetch.time`	The time this statement has spent in other fetches.	msecs
`SQLText.value`	The SQL being executed.	

See Also: "Setting the OC4J JDBC DMS Statement Metrics Option" on page 6-9

OC4J Metrics

This section lists the OC4J J2EE application related metrics.

This section covers the following metrics:

- Web Module Metrics
- Web Context Metrics
- OC4J Servlet Metrics
- OC4J JSP Metrics
- OC4J EJB Metrics
- OC4J OPMN Info Metrics

Web Module Metrics

There is one set of metrics for each Web module within each J2EE application.

The metric table name is `oc4j_web_module`.

Table A–16 OC4J/`application`/WEBs Metrics

Metric	Description	Unit
`parseRequest.active`	Current number of threads trying to read / parse AJP or HTTP requests	
`parseRequest.avg`	Average time spent to read / parse requests	msecs
`parseRequest.completed`	Number of web requests that have been parsed	ops
`parseRequest.maxActive`	Maximum number of threads trying to read / parse AJP or HTTP requests	threads
`parseRequest.maxTime`	Maximum time spent to read / parse requests	msecs
`parseRequest.minTime`	Minimum time spent to read / parse requests	msecs
`parseRequest.time`	Total time spent to read / parse requests from the socket	msecs
`processRequest.active`	Current number of threads servicing web requests	
`processRequest.avg`	Average time spent servicing web requests	msecs
`processRequest.completed`	Number of web requests processed by this application	ops
`processRequest.maxActive`	Maximum number of threads servicing web requests	threads
`processRequest.maxTime`	Maximum time spent servicing a web request	msecs
`processRequest.minTime`	Minimum time spent servicing a web request	msecs
`processRequest.time`	Total time spent servicing this application’s web requests	msecs

Table A–16 (Cont.) OC4J/*application*/WEBs Metrics

Metric	Description	Unit
resolveContext.active	Current number of threads trying to create / find the servlet context	
resolveContext.avg	Average time spent to create / find the servlet context	msecs
resolveContext.completed	Count of completed context resolves	ops
resolveContext.maxActive	Maximum number of threads trying to create / find the servlet context	threads
resolveContext.maxTime	Maximum time spent to create / find the servlet context	msecs
resolveContext.minTime	Minimum time spent to create / find the servlet context	msecs
resolveContext.time	Total time spent to create / find the servlet context. Each web module (WAR) maps to a servlet context	msecs

Web Context Metrics

There is one set of metrics for each Web context module within each J2EE application.

The metric table name is `oc4j_context`.

Table A–17 OC4J/*application*/WEBs/*context* Metrics

Metric	Description	Unit
resolveServlet.time	Total time spent to create / locate servlet instances (within the servlet context)	msecs
resolveServlet.completed	Total Number of lookups for a servlet by OC4J	ops
resolveServlet.minTime	Minimum time spent to create / locate the servlet instance (within the servlet context)	msecs
resolveServlet.maxTime	Maximum time spent to create / locate the servlet instance (within the servlet context)	msecs
resolveServlet.avg	Average time spent to create / locate the servlet instance (within the servlet context)	msecs
sessionActivation.active	Number of active sessions	ops
sessionActivation.time	Total time in which sessions have been active	msecs
sessionActivation.completed	Number of session activations	ops
sessionActivation.minTime	Minimum time a session was active	ops
sessionActivation.maxTime	Maximum time a session was active	msecs
sessionActivation.avg	Average session lifetime	msecs
service.time	Total time spent servicing requests	msecs
service.completed	Total number of requests serviced	ops

Table A–17 (Cont.) OC4J/*application*/WEBs/*context* Metrics

Metric	Description	Unit
`service.minTime`	Minimum time spent servicing requests	msecs
`service.maxTime`	Maximum time spent servicing requests	ops
`service.avg`	Average time spent in servicing the servlet	msecs
`service.active`	Current number of requests active	msecs

OC4J Servlet Metrics

There is one set of metrics for each servlet in each Web module within each J2EE application.

The metric table name is `oc4j_servlet`.

Table A–18 OC4J/*application*/WEBs/*context* /SERVLETS/*servlet* Metrics

Metric	Description	Unit
`service.active`	Current number of threads servicing this servlet	threads
`service.avg`	Average time spent in servicing the servlet	msecs
`service.completed`	Total number of calls to service()	
`service.maxActive`	Maximum number of threads servicing this servlet	threads
`service.maxTime`	Maximum time spent on a servlet's service() call	ops
`service.minTime`	Minimum time spent on a servlet's service() call	msecs
`service.time`	Total time spent on the servlet's service() call	msecs

OC4J JSP Metrics

JSP Runtime Metrics

There is one set of metrics for each Web context for each J2EE application.

The metric table name is `oc4j_jspExec`.

Table A–19 OC4J/*application*/WEBs/*context* /JSP Metrics

Metric	Description	Unit
processRequest.time	Time spent processing requests for JSPs Only used for Context / Application name	msecs
processRequest.completed	Number of requests for JSPs processed by this application	ops
processRequest.minTime	Minimum time spent processing requests for JSPs	msecs
processRequest.maxTime	Maximum time spent processing requests for JSPs	msecs
processRequest.avg	Average time spent processing requests for JSPs	msecs
processRequest.active	Current number of active requests for JSPs	ops

JSP Metrics

There is one set of metrics for each JSP in each Web module.

The metric table names are `oc4j_jsp(threadsafe=true)` and `oc4j_jsp(threadsafe=false)`.

To list these metrics using `dmstool`, enclose the metric table name in quotation marks.

For example:

```
dmstool -table "oc4j_jsp(threadsafe=true)"
```

Table A–20 OC4J/*application*/WEBs/*context* /JSP*jsp_name* Metrics

Metric	Description	Unit
activeInstances.value	Number of active instances. Only used when threadsafe=false	instances
availableInstances.value	Number of available (that is, created) instances. This value is only provided when threadsafe=false.	instances
service.active	Current number of active requests for the JSP	
service.avg	Average time spent servicing the JSP	msecs
service.completed	Number of requests for JSPs processed by this JSP	ops
service.maxTime	Maximum time spent servicing the JSP	msecs
service.minTime	Minimum time spent servicing the JSP	msecs
service.time	Time to serve a JSP (that is, actual execution time of the JSP)	msecs

OC4J EJB Metrics

OC4J EJB Session Bean Metrics

The oc4j_ejb_session_bean metric table includes information on a session bean.

Table A–21 OC4J EJB Session Bean Metrics

Metric	Description	Unit
session-type.value	Provides information on the session type: Stateless or Stateful	String
transaction-type.value	Provides information on the transaction type: Container or	String

EJB Bean Metrics

Oracle Application Server provides a set of these metrics for each type of bean in each EJB jar file in each J2EE application.

The metric table name is oc4j_ejb_entity_bean.

Table A–22 OC4J/application/EJBs/ejb-jar-module/ejb-name Metrics

Metric	Description	Unit
transaction-type.value	Possible values: container or bean	
session-type.value	Possible values: stateful or stateless	
bean-type.value	Possible values: session or entity bean	
exclusive-write-access.value	Possible values: true or false	
isolation.value	Possible values: serializable, uncommitted, committed, repeatable_read, none, DB-determined The value is DB-determined when the isolation attribute is omitted.	
persistence-type.value	Possible values: bean or container	

EJB Method Metrics

There is one set of metrics for each method within each type of EJB bean.

The metric table name is oc4j_ejb_method.

The `client.*` metrics show values for the actual implementation of the method. The `wrapper.*` metrics show values for the wrapper that was automatically generated for the method.

> **See Also:** Chapter 6, "Advanced EJB Subjects" in *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide* for information on automatically generated wrappers.

Table A–23 OC4J/ *`application`* ***/EJBs/*** *`ejb-jar-module`* ***/*** *`ejb-name`* ***/*** *`method-name`* ***Metrics***

Metric	Description	Unit
`client.active`	Current number of threads accessing the actual implementation of this method	ops
`client.avg`	Average time spent inside the actual implementation of this method	msecs
`client.completed`	Number of requests for beans processed by this application	ops
`client.maxActive`	Maximum number of threads accessing the actual implementation of this method	ops
`client.maxTime`	Maximum time spent inside the actual implementation of this method	msecs
`client.minTime`	Minimum time spent inside the actual implementation of this method	msecs
`client.time`	Time spent inside the actual implementation of this method	msecs
`ejbPostCreate.active`	Current number of threads executing `ejbPostCreate`	ops
`ejbPostCreate.avg`	Average time spent in `ejbPostCreate`	msecs
`ejbPostCreate.completed`	Number of times this `ejbPostCreate` has been called	ops
`ejbPostCreate.maxTime`	Maximum time spent in `ejbPostCreate`	msecs
`ejbPostCreate.minTime`	Minimum time spent in `ejbPostCreate`	msecs
`ejbPostCreate.time`	Time spent in the `ejbPostCreate` method (entity beans)	msecs
`trans-attribute.value`	Transaction attribute. Possible values: `NotSupported`, `Supports`, `RequiresNew`, `Mandatory`, and `Never`	
`wrapper.active`	Current number of threads accessing the automatically generated wrapper method	

Table A–23 (Cont.) OC4J/*application*/EJBs/*ejb-jar-module*/*ejb-name*/*method-name* Metrics

Metric	Description	Unit
wrapper.avg	Average time spent inside the automatically generated wrapper method	msecs
wrapper.completed	Number of requests for beans processed by this application	ops
wrapper.maxActive	Maximum number of threads that access the wrapper	ops
wrapper.maxTime	Maximum time spent inside the automatically generated wrapper method	msecs
wrapper.minTime	Minimum time spent inside the automatically generated wrapper method	msecs
wrapper.time	Time spent inside the automatically generated wrapper method. Note: Not all the wrapper methods invoke the actual bean implementation at runtime (for example, create method in a stateless bean). This means that the time spent in the wrapper code could be less than the time spent in the bean implementation	msecs

OC4J OPMN Info Metrics

Table A–24 shows the OC4J OPMN information metrics. The metric table type is `oc4J_opmn`.

Table A–24 OC4J OPMN Information Metrics

Metric	Description	Unit
default_application_log.value	Specifies the default application log file path.	
ias_cluster.value	Specifies the Oracle Application Server cluster name.	String
ias_instance.value	Specifies the Oracle Application Server instance name.	String
jms_log.value	Specifies the JMS log file path.	String
oc4j_instance.value	Specifies the OC4J instance ID.	String
oc4j_island.value	Specifies the OC4J island ID.	String
opmn_group.value	Specifies the OPMN group ID.	String
opmn_sequence.value	Specifies the OPMN sequence ID.	String
rmi_log.value	Specifies the RMI log file path name.	String
server_log.value	Specifies the application server log file path.	String

OC4J JMS Metrics

OC4J JMS metrics are organized into metric tables and fall into two categories:

- JMS API-level metrics: collected on objects visible to the JMS API (for example, connections, sessions, producers, consumers, and browsers). JMS API-level metrics are collected and maintained only for Web and EJB clients (application clients also collect API-level metrics, but do so in their own JVM; these metrics are not available on the OC4J JMS server).
- JMS Server-level metrics: collected by the OC4J JMS server and maintained independent of client-state. JMS Server-level metrics are collected and maintained for all types of clients: Application, Web, and EJB.

Each OC4J JMS metric table (metric table type) contains metrics for instances of the same type; different instances have unique names. For each instance in a metric table, a set of metrics is collected. The names for metrics in each instance are unique IDs that OC4J JMS generates.

Instances may have one or more metrics whose value is the name of another metric instance. For example, the JMS session instances contain metrics that point to the parent containing JMS connection instance. You can use the pointers to navigate through the metrics.

A parent metric instance usually includes a counter metric indicating the number of child metrics of a certain type that have been created. Child metric instances may appear and disappear as the underlying objects are created and destroyed; the counter keeps track of the total number of such instances that were created during the lifetime of the parent.

JMS Metric Tables

OC4J JMS metrics are divided into three types, based on how they are updated:

1. **CTOR** Metrics: Metrics that are set in the constructor or initialization routine of the associated JMS object, and are never changed during the lifetime of the object.
2. **Normal** Metrics: Object level state metrics that are updated as soon as the associated state of the JMS object changes.
3. **Lazy** Metrics: these state metrics are updated lazily, that is, not as soon as the underlying metric value changes, but only periodically (these are typically server store metrics and are updated each time the store is cleaned up of expired messages).

Table A–25 shows a summary of the organization of the OC4J JMS metric tables.

Table A–25 OC4J JMS Metric Tables

JMS Metric Table Type	Parent Table Type	Number of Instances	Description
JMSStats	none	1	Statistics for the OC4J JMS Server
JMSRequestHandlerStats	JMSStats	1 per remote JMS connection	Statistics for the request handler thread servicing a remote JMS connection.
JMSConnectionStats	JMSStats	1 per JMS connection	Statistics for the JMS connections active in this server
JMSSessionStats	JMSConnectionStats	1 per JMS session	Statistics for the JMS sessions active in this server
JMSMessageProducerStats	JMSSessionStats	1 per JMS message producer	Statistics for the JMS producers active in this server
JMSMessageBrowserStats	JMSSessionStats	1 per JMS queue browser	Statistics for the JMS queue browsers in this server
JMSMessageConsumerStats	JMSSessionStats	1 per JMS message consumer	Statistics for the JMS consumers active in this server
JMSDurableSubscriberStats	JMSStats	1 per JMS durable subscriber	Statistics for each JMS durable subscription known to this server
JMSDestinationStats	JMSStats	1 per permanent JMS destination	Statistics for each permanent JMS destination known to the OC4J JMS server
JMSTemporaryDestinationStats	JMSStats	1 per temporary JMS destination	Statistics for each temporary JMS destination known to the OC4J JMS server
JMSStoreStats	JMSDestinationStats JMSTemporaryDestinationStats	1 per server-side message store	Statistics for each message store (one per queue, one per subscription per topic) on the OC4J JMS server
JMSPersistenceStats	JMSDestinationStats	1 per server-side persistent destination	Statistics for operations on the persistence file for each persistent destination

JMS Stats Metric Table

Table A–26 shows the JMS Stats metrics.

The metric table type is `JMSStats`.

Table A–26 JMSStats Metric Table

Metric	Description	Update	Unit
`address.value`	The hostname(s) from which the JMS server accepts remote connections	ctor	string
`connections.count`	The number of JMS connections (local and remote) created by the JMS server	normal	ops
`host.value`	The explicit hostname on which the OC4J JMS server is running.	ctor	string
`oc4j.jms.computeMsgsize.value`	Value of the `oc4j.jms.computeMsgSize` OC4J JMS control knob.	ctor	int
`oc4j.jms.debug.value`	Value of the `oc4j.jms.debug` OC4J JMS control knob	ctor	bool
`oc4j.jms.doGc.value`	Value of the `oc4j.jms.doGc` OC4J JMS control knob	ctor	bool
`oc4j.jms.expirationInterval`	Value of the `oc4j.jms.expirationInterval` OC4J JMS control knob	ctor	msecs
`oc4j.jms.forceRecovery.value`	Value of the `oc4j.jms.forceRecovery` OC4J JMS control knob	ctor	bool
`oc4j.jms.intraSession.value`	Value of the `oc4j.jms.intraSession` OC4J JMS control knob	ctor	bool
`oc4j.jms.j2ee14.value`	Value of the `oc4j.jms.j2ee14` OC4J JMS control knob	ctor	bool
`oc4j.jms.lazySync.value`	Value of the `oc4j.jms.lazySync` OC4J JMS control knob	ctor	bool
`oc4j.jms.listenerAttempts.`	Value of the `oc4j.jms.listenerAttempts` OC4J JMS control knob	ctor	int
`oc4j.jms.maxOpenFiles.value`	Value of the `oc4j.jms.maxOpenFiles` OC4J JMS control knob	ctor	int
`oc4j.jms.messagePoll.value`	Value of the `oc4j.jms.messagePoll` OC4J JMS control knob	ctor	msecs
`oc4j.jms.noDms.value`	Value of the `oc4j.jms.noDms OC4J` JMS control knob	ctor	bool
`oc4j.jms.pagingThreshold.`	Value of the `oc4j.jms.pagingThreshold` OC4J JMS control knob	ctor	double in the range 0.0 to 1.0
`oc4j.jms.saveAllExpired.val`	Value of the `oc4j.jms.saveAllExpired` OC4J JMS control knob	ctor	bool
`oc4j.jms.serverPoll.value`	Value of the `oc4j.jms.serverPoll OC4J` JMS control knob	ctor	msecs
`oc4j.jms.socketBufsize.val`	Value of the `oc4j.jms.socketBufsize` OC4J JMS control knob	ctor	int
`oc4j.jms.usePersistence.val`	Value of the `oc4j.jms.usePersistence` OC4J JMS control knob	ctor	bool
`oc4j.jms.useSockets.value`	Value of the `oc4j.jms.useSockets` OC4J JMS control knob	ctor	bool

Table A–26 (Cont.) JMSStats Metric Table

Metric	Description	Update	Unit
oc4j.jms.useUUID.value	Value of the oc4j.jms.useUUID OC4J JMS control knob	ctor	bool
port.value	The TCP/IP port on which the JMS server listens for incoming connections	ctor	int
requestHandlers.count	The number of request handlers created by the JMS server	normal	int
startTime.value	System.currentTimeMillis() when the OC4J JMS server was started	ctor	msecs
taskManagerInterval.value	The scheduling interval of the OC4J task manager (and the scheduling interval for the OC4J JMS expiration task)	ctor	msecs
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the OC4J JMS server	normal	

JMS Request Handler Stats

Table A–27 shows the JMS Request Handler Stats.

The metric table name is JMSRequestHandlerStats.

Table A–27 JMSRequestHandlerStats Metrics

Metric	Description	Update	Unit
address.value	The hostname from which the remote connection originates (may be an implicit, special address)	ctor	string
connectionID.value	The ID of the JMSConnectionStats instance	ctor	string
host.value	The explicit hostname from which the remote connection originates	ctor	string
port.value	The TCP/IP port from which the remote connection originates	ctor	int
startTime.value	System.currentTimeMillis() when the request handler was started	ctor	string

JMS Connection Stats

Table A–28 shows the JMS Connection Stats.

The metric table name is JMSConnectionStats.

Table A–28 JMSConnectionStats Metrics

Metric	Description	Update	Unit
address.value	The implicit hostname of the remote JMS server host for this connection as specified in the connection factory used to create this connection; set only for non-local connections.	ctor	string
clientID.value	The administratively configured (for ctor) or programmatically set (for normal) clientID for this connection	ctor / normal	string
domain.value	The JMS domain ("queue", "topic", or "unified") of this connection	ctor	string
exceptionListener .value	The stringified name of the current exception listener for this connection	normal	string
host.value	The explicit hostname of the remote JMS server host for this connection; set only for non-local connections	ctor	string
isLocal.value	"true" if and only if the JMS connection is local to the OC4J JMS server in the same JVM	ctor	boolean
isXA.value	"true" if and only if the connection is in XA mode	ctor	boolean
port.value	The remote JMS server port for this connection; set only for non-local connections	ctor	int
startTime.value	System.currentTimeMillis() when this connection was created	ctor	msecs
user.value	The user identity for this connection	ctor	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in this connection object.	normal	

JMS Session Stats

Table A–29 shows the JMS Session Stats.

The metric table name is JMSSessionStats.

Table A–29 JMSSessionStats Metrics

Metric	Description	Update	Unit
acknowledgeMode.value	The acknowledge mode of this session. The valid modes are: AUTO_ACKNOWLEDGE, CLIENT_ACKNOWLEDGE, DUPS_OK_ACKNOWLEDGE, and SESSION_TRANSACTED.	ctor	string
domain.value	The JMS domain ("queue", "topic", or "unified") of this session	ctor	string
isXA.value	"true" if and only if the session is in XA mode	ctor	boolean
sessionListener.value	The stringified name of the current distinguished listener for this session	normal	string
startTime.value	System.currentTimeMillis() when this session was created	ctor	msecs
transacted.value	"true" if and only if the session is transacted	ctor	boolean

Table A–29 (Cont.) JMSSessionStats Metrics

Metric	Description	Update	Unit
`txid.value`	The integer count of the current local transaction associated with this session; the counter is increment each time a local transaction is committed / rolledback; not set for non-transacted session	normal	int
`xid.value`	The Xid of the current distributed transaction associated with this session; set to a null / empty string when in a local transaction mode; not set if the session never participates in a global transaction	normal	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in this session object	normal	

JMS Message Producer Stats

Table A–30 shows the JMS Producer Stats.

The metric table name is `JMSProducerStats`.

Table A–30 JMSProducerStats Metrics

Metric	Description	Update	Unit
`deliveryMode.value`	The current delivery mode of this producer. The valid delivery mode values are: `PERSISTENT` and `NON_PERSISTENT`.	normal	string
`destination.value`	The name of the identified destination for this producer; null / empty for an unidentified producer	ctor	string
`disableMessageID.value`	The value is `true` when message IDs are disabled for the producer	normal	boolean
`disableMessageTimestamp.value`	The value is `true` when message timestamps are disabled for the producer	normal	boolean
`domain.value`	The JMS domain ("queue", "topic", or "unified") of this producer	ctor	string
`priority.value`	The current priority of this producer	normal	int
`startTime.value`	`System.currentTimeMillis()` when this producer was created	ctor	msecs
`timeToLive.value`	The current timeToLive of this producer	normal	msecs
method-name	A phase timer (PhaseEvent Sensor) metric for every major method call in this producer object	normal	

JMS Message Browser Stats

Table A–31 shows the JMS Browser Stats.

The metric table name is `JMSBrowserStats`.

Table A–31 JMSBrowserStats Metrics

Metric	Description	Update	Unit
`destination.value`	The name of the destination for this browser	ctor	string
`selector.value`	The message selector for this browser; null/empty string if unspecified	ctor	string
`startTime.value`	`System.currentTimeMillis()` when this browser was created	ctor	msecs
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in this browser object; calls to "hasMoreElements" and "nextElement" are made on individual enumeration objects, but counted as PhaseEvents in the browser object to simplify data collection, multiple enumerations can be active on the same browser	normal	

JMS Message Consumer Stats

Table A–32 shows the JMS Message Consumer Stats.

The metric table name is JMSMessageConsumerStats.

Table A–32 JMSMessageConsumerStats

Metric	Description	Update	Unit
`destination.value`	The name of the destination for this consumer	ctor	string
`domain.value`	The JMS domain ("queue", "topic", or "unified") of this consumer	ctor	string
`messageListener.value`	The stringified name of the current message listener for this consumer	normal	string
`name.value`	The name of the durable subscriber for this consumer; set only for durable topic subscriptions	ctor	string
`noLocal.value`	The noLocal setting of a subscription; set only for topic consumers	ctor	boolean
`selector.value`	The message selector for this consumer; null/empty string if unspecified	ctor	string
`startTime.value`	`System.currentTimeMillis()` when this consumer was created	ctor	msecs
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in this consumer object	normal	

JMS Durable Subscription Stats

Table A–33 shows the JMS Durable Subscription Stats.

The metric table name is `JMSDurableSubscriptionStats`.

Table A–33 JMSDurableSubscriptionStats Metrics

Metric	Description	Update	Unit
`clientID.value`	The clientID associated with this durable subscriptions	ctor	string
`destination.value`	The name of the topic for this durable subscription	ctor	string
`isActive.value`	"true" if and only if the durable subscription is currently active (being used by a consumer)	normal	boolean
`name.value`	The user-provided name of the durable subscription	ctor	string
`noLocal.value`	The noLocal flag for this durable subscription	ctor	boolean
`selector.value`	The JMS message selector for this durable subscription	ctor	string

JMS Destination Stats

Table A–34 shows the JMS Destination Stats metrics

The metric table name is `JMSDestinationStats`.

Table A–34 JMSDestinationStats Metrics

Metric	Description	Update	Unit
`domain.value`	JMS domain, "queue"or "topic", of the destination	ctor	string
`name.value`	The configured name of the destination. As defined in `jms.xml`	ctor	string
`locations.value`	A comma-separated list of JNDI names bound to the destination. As defined in `jms.xml`	ctor	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the destination object	normal	

JMS Temporary Destination Stats

Table A–35 shows the JMS Temporary Destination Stats.

The metric table name is `JMSTempoaryDestinationStats`.

Table A–35 JMSTemporaryDestinationStats Metrics

Metric	Description	Update	Unit
`connectionID.value`	The ID of the JMSConnectionStats instance from which this temporary destination was created	ctor	string
`domain.value`	JMS domain, for example "queue" or "topic", of the destination	ctor	string
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the destination object	normal	

JMS Store Stats

Table A–36 shows the JMS StoreStats metric table.

The metric table name is `JMSStoreStats`.

Table A–36 JMSStoreStats Metric

Metric	Description	Update	Unit
`destination.value`	A pretty-printed name of the JMS destination associated with this message store	ctor	string
`messageCount.value`	Total number of messages contained in this store	lazy	int
`messageDequeued.count`	Total number of message dequeues (transacted or otherwise)	normal	ops
`messageDiscarded.count`	Total number of message discarded after the rollback of an enqueue	normal	ops
`messageEnqueued.count`	Total number of message enqueues (transacted or otherwise)	normal	ops
`messageExpired.count`	Total number of message expirations	normal	ops
`messagePagedIn.count`	Total number of message bodies paged in	normal	ops
`messagePagedOut.count`	Total number of message bodies paged out store (modulo the setting of the	normal	ops
`messageRecovered.count`	Total number of messages recovered (either from a persistence file, or after the rollback of a dequeue)	normal	ops
`pendingMessageCount.value`	Total number of messages part of an enqueue / dequeue of an active transaction	lazy	int
`storeSize.value`	Total size, in bytes, of the message `oc4j.jms.computeMsgsize` control knob	lazy	bytes
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the message store object	normal	

The following identity holds:

```
messageCount = messageRecovered + messageEnqueued -
messageDequeued - messageDiscarded - messageExpired
```

If a message is both enqueued and dequeued in the same transaction, the `messageEnqueued` and `messageDequeued` events occur, but the `messageRecovered` and `messageDiscarded` events do not.

JMS Persistence Stats

Table A–37 shows the JMS Persistence Stats.

The metric table name is `JMSPersistenceStats`.

Table A–37 JMSPersistenceStats Metrics

Metric	Description	Update	Unit
`destination.value`	A pretty-printed name for the JMS destination associated with this persistence file	ctor	string
`holePageCount.value`	The number of 512b pages currently free in this file	normal	int
`isOpen.value`	"true" iff the persistence file descriptor is currently open (for LRU caching)	normal	boolean
`lastUsed.value`	`System.currentTimeMillis()` when this persistence file was last used (for LRUcaching)	normal	msecs
`persistenceFile.value`	The absolute path name of the persistence file used for this persistent destination. This value differs depending on the operating system where OC4J is running.	ctor	string
`usedPageCount.value`	The number of 512b pages currently in use in this file	normal	int
method-name	An interval timer metric (PhaseEvent Sensor) for every major method call in the persistence file object	normal	

OC4J Task Manager Metrics

The metric table type is `oc4j_task`.

Table A–38 OC4J_taskManager Metrics

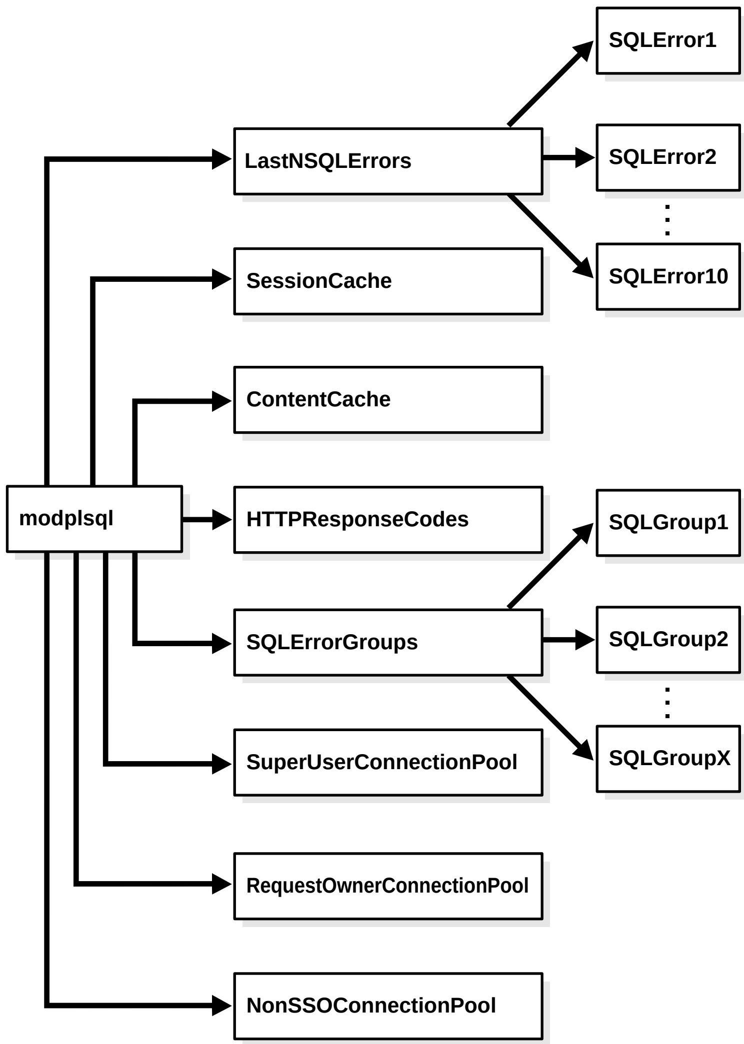
Metric	Description	Unit
`interval.value`	Shows how often the task should run. The task manager executes all the tasks in a round-robin fashion. If the interval is zero, then the task manager executes the task when it is selected in the round robin.	msecs (Milliseconds)
`run().active`	Number of active threads.	threads
`run().avg`	Average time for the taskmanager to run the task	msecs
`run().completed`	Number of times the taskmanager has run the task.	ops
`run().maxActive`	Maximum number of active tasks.	threads
`run().maxTime`	Maximum time for the task to run.	msecs
`run().minTime`	Minimum time for the task to run.	msecs
`run().time`	Total time spent running the task manager	msecs

mod_plsql Metrics

This section describes the Oracle Application Server mod_plsql metrics.

Figure A–1, "mod_plsql Metric Tree" shows the structure of the mod_plsql metrics. The tables in this section describe the relevant metrics.

Figure A–1 mod_plsql Metric Tree



The `/modplsql/HTTPResponseCodes` Metrics lists the response codes returned by `mod_plsql`.

The metric table name is `modplsql_HTTPResponseCodes`. This metric table includes one metric containing the count, number of times the response was generated, for each HTTP response type.

```
[type=modplsql_HTTPResponseCodes]
```

For example, the `http404.count` metric holds a count of the "HTTP 404: Not found" response codes.

Table A–39 lists the set of metrics for the `mod_plsql` session cache.

The metric table name is `modplsql_Cache`.

Table A–39 mod_plsql/SessionCache Metrics

Metric	Description	Unit
`cacheStatus.value`	Status of the cache. This can be either enabled or disabled.	status
`newMisses.count`	Number of session cache misses (new)	ops
`staleMisses.count`	Number of session cache misses (stale)	ops
`hits.count`	Number of session cache hits	ops
`requests.count`	Number of requests to the session cache	ops

Table A–40 lists the set of metrics for the `mod_plsql` content cache.

The metric table name is `modplsql_ContentCache`.

Table A–40 mod_plsql/ContentCache Metrics

Metric	Description	Unit
`cacheStatus.value`	Status of the cache, either enabled or disabled.	
`newMisses.count`	Number of content cache misses (new)	ops
`staleMisses.count`	Number of content cache misses (stale)	ops
`hits.count`	Number of content cache hits	ops
`requests.count`	Number of requests to the content cache	ops

The `SQLErrorGroups` metrics show the predefined groupings of SQL errors. For each group, the metrics in Table A–41 are recorded.

The metric table name is `modplsql_SQLErrorGroup`:

```
/modplsql/SQLErrorGroups/group [type=modplsql_SQLErrorGroup]
```

The *group* is based on the groupings in the Oracle SQL error documentation. For example, the metric name `Ora24280Ora29249` represents the grouping Ora-24280 to Ora-29249. Each SQL error that occurs as a result of executing a request is put into the appropriate group based on its error code. If you are getting a high number of the same errors, you should investigate what is causing the problem, using the Oracle SQL error message documentation for further details on the error message.

Table A–41 mod_plsql/SQLErrorGroups Metrics

Metric	Description	Unit
`lastErrorDate.value`	Date of the last request to cause the SQL error	date
`lastErrorRequest.value`	Last request to cause the SQL error	url
`lastErrorText.value`	SQL error text of the last error	error
`error.count`	Number of errors that have occurred within the group	ops

The `LastNSQLErrors` statistics show the last 10 SQL errors that have occurred while executing requests. These are updated in a round robin fashion. For each error, the metrics in Table A–42 are recorded.

The metric table name is `modplsql_LastNSQLError`:

/modplsql/LastNSQLErrors/<*SQL Error Slot*> [type=modplsql_LastNSQLError]

If you are getting a large number of the same errors, you should investigate what is causing the problem. Refer to the Oracle SQL error messages documentation for further details of the error represented by the `errorText.value` metric.

Table A–42 mod_plsql/LastNSQLErrors Metrics

Metric	Description	Unit
`errorDate.value`	Date the request caused the SQL error	date
`errorRequest.value`	Request causing the SQL error	url
`errorText.value`	SQL error text	error

Table A–43 lists the set of metrics for the Non-SSO connection pool.

The metric table name is `modplsql_DatabaseConnectionPool`:

```
/modplsql/NonSSOConnectionPool [type=modplsql_DatabaseConnectionPool]
```

Table A–43 mod_plsql/NonSSOConnectionPool Metrics

Metric	Description	Unit
connFetch.maxTime	Maximum time to fetch a connection from the pool	usecs
connFetch.minTime	Minimum time to fetch a connection from the pool	usecs
connFetch.avg	Average time to fetch a connection from the pool	usecs
connFetch.active	Child servers currently in the pool fetch phase	threads
connFetch.time	Total time spent fetching connections from the pool	usecs
connFetch.completed	Number of times a connection has been requested from the pool	ops
newMisses.count	Number of connection pool misses (new)	ops
staleMisses.count	Number of connection pool misses (stale)	ops
hits.count	Number of connection pool hits	ops

Table A–44 lists the set of metrics for the request owner connection pool.

The metric table name is `modplsql_DatabaseConnectionPool`:

`/modplsql/RequestOwnerConnectionPool [type=modplsql_DatabaseConnectionPool]`

Table A–44 mod_plsql/RequestOwnerConnectionPool Metrics

Metric	Description	Unit
connFetch.maxTime	Maximum time to fetch a connection from the pool	usecs
connFetch.minTime	Minimum time to fetch a connection from the pool	usecs
connFetch.avg	Average time to fetch a connection from the pool	usecs
connFetch.active	Child servers currently in the pool fetch phase	threads
connFetch.time	Total time spent fetching connections from the pool	usecs
connFetch.completed	Number of times a connection has been requested from the pool	ops
newMisses.count	Number of connection pool misses (new)	ops
staleMisses.count	Number of connection pool misses (stale)	ops
hits.count	Number of connection pool hits	ops

Table A–45 lists the set of metrics for the super user connection pool.

The metric table name is `modplsql_DatabaseConnectionPool`:

`/modplsql/SuperUserConnectionPool [type=modplsql_DatabaseConnectionPool]`

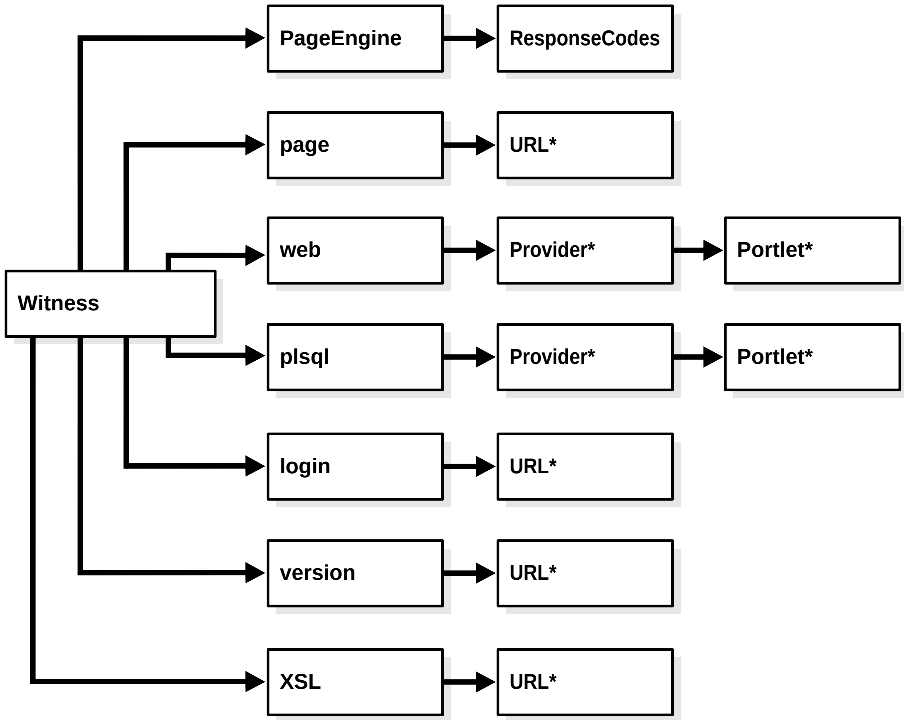
Table A–45 mod_plsql/SuperUserConnectionPool Metrics

Metric	Description	Unit
connFetch.maxTime	Maximum time to fetch a connection from the pool	usecs
connFetch.minTime	Minimum time to fetch a connection from the pool	usecs
connFetch.avg	Average time to fetch a connection from the pool	usecs
connFetch.active	Threads currently in the pool fetch phase	threads
connFetch.time	Total time spent fetching connections from the pool	usecs
connFetch.completed	Number of times a connection has been requested from the pool	ops
newMisses.count	Number of connection pool misses (new)	ops
staleMisses.count	Number of connection pool misses (stale)	ops
hits.count	Number of connection pool hits	ops

Portal Metrics

This section shows the Portal Metrics. Figure A–2, "Parallel Page Engine Metric Tree" shows the structure of the Portal Parallel Page Engine metrics. The tables in this section describe the relevant metrics.

Figure A–2 Parallel Page Engine Metric Tree



The set of metrics can be broken down into static and dynamic types. Static metrics are those that are always available and dynamic being those metrics that only appear if a specific event occurs, such as when a specific portlet is requested. All of the `PageEngine` and `ResponseCodes` metrics are static, the remaining metrics are dynamic.

Table A–46 lists the set of metrics for the Parallel Page Engine. The metric table type is `modplsql_PageEngine`. This set represents the general performance of the Parallel Page Engine. If you intend to use the cache you should ensure that the `cacheEnabled.value` metric is set 1. To turn the cache on, refer to the `mod_plsql` cache and Parallel Page Engine configuration documentation.

Table A–46 Witness/PageEngine Metrics

Metric	Description	Unit
pageRequests.value	Total number of requests for Portal pages.	count
cacheEnabled.value	The PPE makes use of the mid tier cache as controlled by mod_plsql, and is accessed via a JNI layer. This flag indicates whether this JNI cache as accessed from the PPE is enabled or not. This flag will be zero if the cache is either configured to be off or there was a problem loading the JNI layer DLL.	status
cachePageHits.value	Number of requests for cacheable fully assembled pages that have resulted in a cache hit.	count
cachePageRequests.value	Number of requests for cacheable fully assembled pages.	count
pageMetadataWaitTimeAvg.value	Average time spent in the PPE internal request queue waiting for page metadata, for all requests. To obtain the average you should divide the value metric by the count metric. The value being the accumulative time for all requests and the count being the number of requests made.	msecs
pageMetadataWaitTimeAvg.count	Number of requests made for page metadata. This metric should be used in conjunction with pageMetadataWaitTimeAvg.value to calculate the average time spent in the PPE internal request queue.	ops
pageMetadataWaitTime.value	Time the last page metadata request spent in the PPE internal request queue.	msecs
pageMetadataWaitTime.count	Number of requests for page metadata.	ops
pageMetadataWaitTime.minValue	Minimum time spent in the PPE internal request queue waiting for page metadata to be requested.	msecs
pageMetadataWaitTime.maxValue	Maximum time spent in the PPE internal request queue waiting for page metadata to be requested.	msecs
pageElapsedTimeAvg.value	Average time to generate pages, including fetching the page metadata. To obtain the average you should divide the value metric by the count metric. The value being the accumulative time for all requests and the count being the number of requests made.	msecs
pageElapsedTimeAvg.count	Number of pages that had to be generated (that is, not cached). This metric should be used in conjunction with pageElapsedTimeAvg.value to calculate the average time to generate pages, including fetching the page metadata.	ops
pageElapsedTime.value	Time to generate the last page requested, including fetching the page metadata.	msecs
pageElapsedTime.count	Number of pages that had to be generated (that is, not cached).	ops
pageElapsedTime.minValue	Minimum time to generate a page, including fetching the page metadata.	msecs
pageElapsedTime.maxValue	Maximum time to generate a page, including fetching the page metadata.	msecs
pageMetadataFetchTimeAvg.value	Average time to fetch page metadata, for all requests. To obtain the average you should divide the value metric by the count metric. The value being the accumulative time for all requests and the count being the number of requests made.	msecs

Table A–46 (Cont.) Witness/PageEngine Metrics

Metric	Description	Unit
pageMetadataFetchTimeAvg.count	Number of requests for page metadata. This metric should be used in conjunction with pageMetadataFetchTimeAvg.value to calculate the average time to fetch page metadata.	ops
pageMetadataFetchTime.value	Time to fetch page metadata, for the last request.	msecs
pageMetadataFetchTime.count	Number of requests for page metadata.	ops
pageMetadataFetchTime.minValue	Minimum time to fetch page metadata.	msecs
pageMetadataFetchTime.maxValue	Maximum time to fetch page metadata.	msecs
queueTimeout.value	Number of requests for Portal data that have timed out in the PPE internal request queue.	msecs
queueStayAvg.value	Average time all internal PPE requests spent in the PPE internal request queue. To obtain the average you should divide the value metric by the count metric. The value being the accumulative time for all requests and the count being the number of requests made.	msecs
queueStayAvg.count	Number of requests added to the internal PPE request queue. This metric should be used in conjunction with queueStayAvg.value to calculate the average time requests spent in the internal PPE request queue.	ops
queueStay.value	Time the last internal PPE request spent in the PPE internal request queue.	msecs
queueStay.count	Number of requests added to the internal PPE request queue.	ops
queueStay.minValue	Minimum time a request spent in the internal PPE request queue.	msecs
queueStay.maxValue	Maximum time a request spent in the internal PPE request queue.	msecs
queueLengthAvg.value	Average length of the PPE internal request queue. To obtain the average you should divide the value metric by the count metric.	msecs
queueLengthAvg.count	Number of requests added to the PPE internal request queue. This metric should be used in conjunction with queueLengthAvg.value to calculate the average length of the PPE internal request queue.	ops
queueLength.value	Current length of the PPE internal request queue.	msecs
queueLength.count	Number of requests added to the PPE internal request queue.	ops
queueLength.minValue	Minimum number of requests in the PPE internal request queue.	msecs
queueLength.maxValue	Maximum number of requests in the PPE internal request queue.	msecs

The set of metrics for the response codes returned by internal requests, such as portlets, page, or metadata, made by the Parallel Page Engine are in the metric table is `modplsql_PageEngine_ResponseCodes`.

This table contains a count for each HTTP response type.

For example, `http100.count`, contains a count of the HTTP:100 Continue response codes.

In addition, the metric `httpUnresolvedRedirect.value` contains a count of requests that were not resolved after returning a redirect HTTP response code and `httpTimeout.value` contains a count of requests that timed out in the PPE internal request queue.

Table A–47 lists the set of metrics for the internal Parallel Page Engine page metadata requests. The metric table name is dynamic in that it includes the URL used to request the page metadata. If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–47 Witness/page/`url` Metrics

Metric	Description	Unit
`lastResponseDate.value`	Last time the response was made	Date
`lastResponseCode.value`	Last response code returned for this request	HTTP response code
`cacheHits.value`	Number of cache hits for this request	ops
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently being processed	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–48 lists the set of metrics for the internal Parallel Page Engine login metadata requests. The metric table name is dynamic in that it includes the URL used to request the login metadata. If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–48 Witness/login/`url` Metrics

Metric	Description	Unit
`lastResponseDate.value`	Last time the request was made	Date
`lastResponseCode.value`	Last response code returned for this request	HTTP response code
`cacheHits.value`	Number of cache hits for this request	ops
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–49 lists the set of metrics for the internal Parallel Page Engine Portal version requests. The metric table name is dynamic in that it includes the URL used to request the version of the Portal repository. If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–49 Witness/version/`url` Metrics

Metric	Description	Unit
`lastResponseDate.value`	Last time the request was made	Date
`lastResponseCode.value`	Last response code returned for this request	HTTP response code
`cacheHits.value`	Number of cache hits for this request	ops
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–50 lists the set of metrics for the internal Parallel Page Engine Portal XSL requests. The metric table name is dynamic in that it includes the URL used to request the XSL document. If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–50 Witness/XSL/`url` Metrics

Metric	Description	Unit
`lastResponseDate.value`	Last time the request was made	Date
`lastResponseCode.value`	Last response code returned for this request	HTTP response code
`cacheHits.value`	Number of cache hits for this request	ops
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–51 lists the set of metrics for the internal Parallel Page Engine PL/SQL provider requests, holding a metric summary of all the requested portlets owned by a specific provider. The metric table name is dynamic in that it includes the provider name. `dad-provider` indicates the name of the DAD that the named provider is registered and accessed through. If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–51 Witness/plsql/`dad-provider` Metrics

Metric	Description	Unit
`cacheHits.value`	Number of cache hits for this request	ops
`offline.value`	Flag to indicate whether the provider is offline. A value of 1 indicates that the provider is offline and a value of 0 indicates that the provider is online.	state
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs

Table A–51 (Cont.) Witness/plsql/`dad-provider` Metrics

Metric	Description	Unit
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–52 lists the set of metrics for the internal Parallel Page Engine Portal PL/SQL portlet requests. The metric table name is dynamic in that it includes both the provider and portlet names. Table A–51 contains metrics summarizing all of the portlets requested that are owned by a specific PL/SQL provider.

If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–52 Witness/plsql/`dad-provider`/`portlet` Metrics

Metric	Description	Unit
`lastResponseDate.value`	Last time the request was made	Date
`lastResponseCode.value`	Last response code returned for this request	HTTP response code
`cacheHits.value`	Number of cache hits for this request	ops
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–53 lists the set of metrics for the internal Parallel Page Engine Web provider requests, holding a metric summary of all the requested portlets owned by a specific provider. The metric table name is dynamic in that it includes the provider name. If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details.

Table A–53 Witness/Web/*dad-provider* Metrics

Metric	Description	Unit
`cacheHits.value`	Number of cache hits for this request	ops
`offline.value`	Flag to indicate whether the provider is offline. A value of 1 indicates that the provider is offline and a value of 0 indicates that the provider is online.	state
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

Table A–54 lists the set of metrics for the internal Parallel Page Engine Portal Web portlet requests. The metric name is dynamic in that it includes both the provider and portlet names. Table A–53 contains metrics summarizing all of the portlets requested that are owned by a specific Web provider.

If you are encountering a large number of failed requests, check the HTTPD `error_log` for details of why the requests are failing. The `mod_plsql` metrics may also provide further details. If you are seeing a large number of HTTP redirects (302), consider coding the portlet to avoid the redirect as this helps performance. If you have coded you portlet to be cacheable and the number of cache hits is low, check the `mod_plsql` cache settings to ensure they are set to the appropriate levels for your system.

Table A–54 Witness/Web/*dad-provider*/*portlet* Metrics

Metric	Description	Unit
`lastResponseDate.value`	Last time the request was made	Date
`lastResponseCode.value`	Last response code returned for this request	HTTP response code
`cacheHits.value`	Number of cache hits for this request	ops
`httpXXX.value`	Count of specific HTTP response codes for this request.	ops
`executeTime.maxTime`	Maximum time to make the request	usecs
`executeTime.minTime`	Minimum time to make the request	usecs
`executeTime.avg`	Average time to make the request	usecs

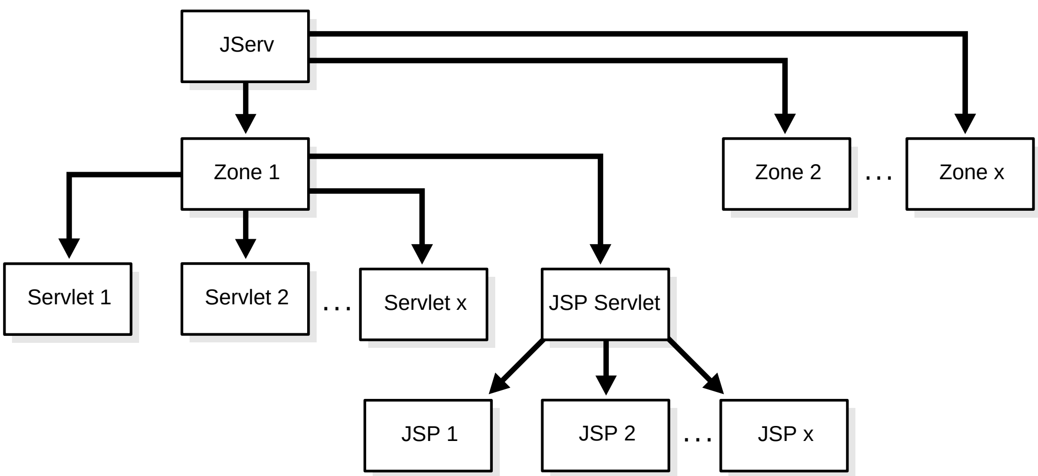
Table A–54 (Cont.) Witness/Web/ *`dad-provider`* ***/*** *`portlet`* ***Metrics***

Metric	Description	Unit
`executeTime.active`	Threads currently in the make request phase	threads
`executeTime.time`	Total time spent making requests	usecs
`connFetch.completed`	Number of requests made	ops

JServ Metrics

Figure A–3 shows the structure of the JServ metrics. The following tables describe the relevant metrics.

Figure A–3 JServ Metric Tree



Overall JServ Metrics

There is one set of metrics for each JServ server process.

The metric table name is jserv_server.

Table A–55 jserv Metric Tree

Metric	Description	Unit
`port.value`	The ID of the TCP port on which this JServ listens	
`readRequest.active`	Threads currently in the `readRequest` processing phase	
`readRequest.avg`	Average time to read and parse requests	msecs
`readRequest.maxTime`	Maximum time to read and parse requests	msecs
`readRequest.minTime`	Minimum time to read and parse requests	msecs
`readRequest.completed`	Number of times the `readRequest` processing phase has completed	ops
`readRequest.time`	Total time to read and parse the request	msecs
`maxConnections.value`	Number of requests that can be handled concurrently in the JServ process	threads
`activeConnections.maxValue`	Maximum number of requests being processed simultaneously	threads
`activeConnections.value`	Number of requests being processed simultaneously	threads
`idlePeriod.maxTime`	Maximum time process was not handling any requests	msecs
`idlePeriod.minTime`	Minimum time process was not handling any requests	msecs
`idlePeriod.completed`	Number of times no requests were being serviced	ops
`idlePeriod.time`	Total time process was not handling any requests	msecs
`host.value`	Hostname/IP address this JServ process binds to	
`maxBacklog.value`	Maximum number of backlog requests that may be queued in the OS waiting for this JServ	integer

JServ Zone Metrics

There is one set of metrics for each JServ zone.

The metric table name is jserv_zone.

Table A–56 jserv/zone Metrics

Metric	Description	Unit
`checkReload.active`	Threads currently in the `checkReload` processing phase	integer
`checkReload.avg`	Average time to check if the zone must be reloaded	msecs
`checkReload.maxTime`	Maximum time to check if the zone must be reloaded	msecs
`checkReload.minTime`	Minimum time to check if the zone must be reloaded	msecs
`checkReload.completed`	Number of times the `checkReload` processing phase has completed	ops
`checkReload.time`	Total time to check if the zone must be reloaded	msecs
`activeSessions.value`	The number of sessions which exist in this zone	sessions

Table A–56 (Cont.) jserv/*zone* Metrics

Metric	Description	Unit
`readSession.count`	Number of times session data has been read with `HttpSession.getValue` in this zone	ops
`writeSession.count`	Number of times session data has been written with `HttpSession.putValue` in this zone	ops
`loadFailed.count`	Number of times we failed to load the requested application (does not work for OJSPs)	ops

JServ Servlet Metrics

There is one set of metrics per servlet. Note that the JSP servlet holds all of the aggregated load metrics for all servlets and JSPs within a zone.

The metric table name is jserv_servlet.

Table A–57 /jserv/*zone*/*servlet* Metrics

Metric	Description	Unit
`processRequest.active`	Threads currently in the `processRequest` processing phase	integer
`processRequest.avg`	Average time to completely process servlet (including JServ engine overhead)	msecs
`processRequest.maxTime`	Maximum time to completely process servlet (including JServ engine overhead)	msecs
`processRequest.minTime`	Minimum time to completely process servlet (including JServ engine overhead)	msecs
`processRequest.completed`	Number of times the `processRequest` processing phase has completed	ops
`processRequest.time`	Total time to completely process servlet (including JServ engine overhead)	msecs
`serviceRequest.active`	Threads currently in the `serviceRequest` processing phase	integer
`serviceRequest.avg`	Average time for service method implementing this application (excluding JServ engine overhead)	msecs
`serviceRequest.maxTime`	Maximum time for service method implementing this application (excluding JServ engine overhead)	msecs
`serviceRequest.minTime`	Minimum time for service method implementing this application (excluding JServ engine overhead)	msecs
`serviceRequest.completed`	Number of times the `serviceRequest` processing phase has completed	ops
`serviceRequest.time`	Total time for service method implementing this application (excluding JServ engine overhead)	msecs

Table A–57 (Cont.) /jserv/*zone*/*servlet* Metrics

Metric	Description	Unit
`loadServlet.avg`	Average time to load servlet (from cache or file)	msecs
`loadServlet.maxTime`	Maximum time to load servlet (from cache or file)	msecs
`loadServlet.minTime`	Minimum time to load servlet (from cache or file)	msecs
`loadServlet.completed`	Number of times the `loadServlet` processing phase has completed	ops
`loadServlet.time`	Total time to load servlet (from cache or file)	msecs
`loadServletClasses.active`	Threads currently in the `loadServletClasses` processing phase	integer
`loadServletClasses.avg`	Average time to load servlet classes from file	msecs
`loadServletClasses.maxTime`	Maximum time to load servlet classes from file	msecs
`loadServletClasses.minTime`	Minimum time to load servlet classes from file	msecs
`loadServletClasses.completed`	Number of times the `loadServletClasses` processing phase has completed. For most classes, this value is usually 1	ops
`loadServletClasses.time`	Total time to load servlet classes from file	msecs
`loadServlet.avg`	Average time to load servlet (from cache or file)	msecs
`createSession.active`	Threads currently in the `createSession` processing phase	
`createSession.avg`	Average time to create a session	msecs
`createSession.maxTime`	Maximum time to create a session	msecs
`createSession.minTime`	Minimum time to create a session	msecs
`createSession.completed`	Number of times the `createSession` processing phase has completed number of sessions that have been created for this application	ops
`createSession.time`	Total time to create a session	msecs
`maxSTMInstances.value`	Total number of instances available for this `SingleThreadModel` servlet	integer
`activeSTMInstances.maxValue`	Maximum number of instances concurrently servicing requests for this `SingleThreadModel`	integer
`activeSTMInstances.value`	Total number of instances available for this `SingleThreadModel` servlet	instances

JServ JSP Metrics

There is one set of metrics per JSP. Note that the JSP servlet holds all of the aggregated load metrics for all servlets and JSPs within a zone.

The metric table name is `jserv_jsp`.

Table A–58 /jserv/*zone*/*servlet* Metrics

Metric	Description	Unit
processRequest.active	Threads currently in the processRequest processing phase	integer
processRequest.avg	Average time to completely process servlet (including JServ engine overhead)	msecs
processRequest.maxTime	Maximum time to completely process servlet (including JServ engine overhead)	msecs
processRequest.minTime	Minimum time to completely process servlet (including JServ engine overhead)	msecs
processRequest.completed	Number of times the processRequest processing phase has completed	ops
processRequest.time	Total time to completely process servlet (including JServ engine overhead)	msecs
serviceRequest.active	Threads currently in the serviceRequest processing phase	integer
serviceRequest.avg	Average time for service method implementing this application (excluding JServ engine overhead)	msecs
serviceRequest.maxTime	Maximum time for service method implementing this application (excluding JServ engine overhead)	msecs
serviceRequest.minTime	Minimum time for service method implementing this application (excluding JServ engine overhead)	msecs
serviceRequest.completed	Number of times the serviceRequest processing phase has completed	ops
serviceRequest.time	Total time for service method implementing this application (excluding JServ engine overhead)	msecs
loadServlet.avg	Average time to load servlet (from cache or file)	msecs
loadServlet.maxTime	Maximum time to load servlet (from cache or file)	msecs
loadServlet.minTime	Minimum time to load servlet (from cache or file)	msecs
loadServlet.completed	Number of times the loadServlet processing phase has completed	ops
loadServlet.time	Total time to load servlet (from cache or file)	msecs
loadServletClasses.active	Threads currently in the loadServletClasses processing phase	
loadServletClasses.avg	Average time to load servlet classes from file	msecs
loadServletClasses.maxTime	Maximum time to load servlet classes from file	msecs
loadServletClasses.minTime	Minimum time to load servlet classes from file	msecs
loadServletClasses.completed	Number of times the loadServletClasses processing phase has completed. For most classes, this value is usually 1	ops
loadServletClasses.time	Total time to load servlet classes from file	msecs
loadServlet.avg	Average time to load servlet (from cache or file)	msecs
createSession.active	Threads currently in the createSession processing phase	
createSession.avg	Average time to create a session	msecs

Table A–58 (Cont.) /jserv/zone/servlet Metrics

Metric	Description	Unit
`createSession.maxTime`	Maximum time to create a session	msecs
`createSession.minTime`	Minimum time to create a session	msecs
`createSession.completed`	Number of times the `createSession` processing phase has completed number of sessions that have been created for this application	ops
`createSession.time`	Total time to create a session	msecs
`maxSTMInstances.value`	Total number of instances available for this `SingleThreadModel` servlet	
`activeSTMInstances.maxValue`	Maximum number of instances concurrently servicing requests for this `SingleThreadModel`	
`activeSTMInstances.value`	Total number of instances available for this `SingleThreadModel` servlet	instances

Oracle Process Manager and Notification Server Metrics

This sections lists the Oracle Process Manager and Notification Server (opmn) metrics.

This section includes the following:

- OPMN_PM Metric Table
- OPMN_HOST_STATISTICS Metric Table
- OPMN_IAS_INSTANCE Metric Table
- OPMN_IAS_COMPONENT Metrics
- OPMN ONS Metrics

OPMN_PM Metric Table

The `opmn_pm` metric table is the root of the process manager subtree for the OPMN DMS metrics. The metrics in this metric table contain statistics about OPMN requests. An OPMN request is a command that has been issued to OPMN from a client, for example DCM, to perform an operation on one or more OPMN managed processes.

Requests can have one of three possible results:

- Success – success means OPMN handles the request successfully.

- Partial Success – partial Success means OPMN only handles part of the request successfully. For example, if a client wants OPMN to start three OC4J processes, and only two are successfully started, the request result is partial success.
- Failure – failure means the request fails.

Table A–59 shows the metric table type `opmn_pm`.

Table A–59 OPMN_PM Metrics

Metric	Description	Unit
`jobWorkerQueue.value`	Specifies the number of jobs in the OPMN worker queue	ops
`lReq.count`	Specifies the number of local HTTP requests which OPMN handles	ops
`procDeath.count`	Specifies the number of processes which die after the process manager starts them	ops
`procDeathReplace.count`	Specifies the number of processes which are restarted after the process manager detects they are dead	ops
`reqFail.count`	Specifies the number of HTTP requests which fail	ops
`reqPartialSucc.count`	Specifies the number of HTTP requests which partially succeed	ops
`reqSucc.count`	Specifies the number of HTTP requests which succeed	ops
`rReq.count`	Specifies the number of remote HTTP requests which OPMN handles	ops
`workerThread.value`	Specifies the number of worker threads	threads

OPMN_HOST_STATISTICS Metric Table

The OPMN host statistics metric table provides information on the host running the OPMN process.

Appendix A–60 shows the metric table type `opmn_host_statistics`.

Table A–60 OPMN_HOST_STATISTICS Metrics

Metric	Description	Unit
`cpuIdle.value`	Specifies the number of milliseconds the cpu(s) have been idle since an unspecified time.	milliseconds
`freePhysicalMem.value`	Specifies the amount of free physical memory on the host machine.	kilobytes
`numProcessors.value`	Specifies the number of processors available on the host machine.	integer
`timestamp.value`	Specifies the time that host statistics are taken. The timestamp is the number of milliseconds from an unspecified time.	milliseconds from an unspecified time
`totalPhysicalMem.value`	Specifies the total physical memory available on the host machine.	kilobytes

OPMN_IAS_INSTANCE Metric Table

The OPMN IAS instance subtree shows the Oracle Application Server instance node name.

Table A–61 shows the metric table type `opmn_ias_instance`.

Table A–61 OPMN_IAS_INSTANCE Metrics

Metric	Description	Unit
`iasCluster.value`	Specifies the Oracle Application Server cluster name for the Oracle Application Server instance.	String

OPMN_IAS_COMPONENT Metrics

The OPMN IAS component subtree represents an Oracle Application Server component. The OPMN IAS component subtree includes several metric tables containing component information.

Table A–62 shows the metric table type `opmn_process_type`.

Table A–62 OPMN_PROCESS_TYPE Metrics

Metric	Description	Unit
`moduleId.value`	Specifies the values of attribute module-IDs, as specified in the process-type tag in the `opmn.xml` configuration file.	String

Table A–63 shows the metric table type `opmn_process_set`.

Table A–63 OPMN_PROCESS_SET Metrics

Metric	Description	Unit
`numProcConf.value`	Specifies the number, or maximum number, of processes configured for this process set.	String (integer)
`reqFail.count`	Specifies the number of HTTP requests which fail for this process set.	ops
`reqPartialSucc.count`	Specifies the number of HTTP requests which partially succeed for this process set.	ops
`reqSucc.count`	Specifies the number of HTTP requests which succeed for this process set	ops
`restartOnDeath.value`	Specifies whether, when a process dies, OPMN should restart the process.	String (boolean)

Table A–64 shows the metric table type `opmn_process`.

Table A–64 OPMN_PROCESS Metrics

Metric	Description	Unit
cpuTime.value	Shows the amount of CPU time used by the process.	CPU msecs
heapSize.value	Shows the heap size of the process.	Kilobytes
iasCluster.value	Shows the Oracle Application Server cluster name for the process	String
iasInstance.value	Shows the Oracle Application Server instance name for the process	String
indexInSet.value	Shows the process index in the process set. This value is only valid for OPMN managed processes, for OPMN unmanaged processes, this value has no meaning, and the value is always 0.	String (integer)
memoryUsed.value	The amount of memory used by the process. This metric is calculated in an OS-specific manner. On UNIX, this is the process image memory used value. This is all the memory in use by the process. On Windows, this is the working set memory used value. This is the same value that is reported by the Task Manager under the mem usage column. The working set is the set of memory pages touched recently by the threads in the process. If free memory in the system is above a certain threshold, pages are left in the working set of a process, even if they are not in use. When free memory falls below a certain threshold, pages are trimmed from the working sets. If needed, pages are soft-faulted back into the working set before they leave main memory.	
pid.value	The process ID for the process.	
privateMemory.value	The private memory of the process.	Kilobytes
sharedMemory.value	The shared memory for the process	Kilobytes
startTime.value	The start time of the process.	msecs
status.value	The status of the process. The status can have the following values: ■ NONE – New process slot, no operations have been applied yet (no status). ■ Init – process has been started, opmn is waiting for initialization to complete. ■ Alive – process is fully started. ■ Stop – process stop operation is in progress. ■ Stopped – process has been fully stopped. ■ Bounce – non-terminating process restart is in progress. ■ Restart – process stop operation is in progress, prior to a new start being issued. ■ InitFail – failure before init timeout reached, a stop and start will be attempted in the retry limit has not been reached. ■ BounceFail – non-terminating process restart failed, as stop and start will be attempted if the retry limit has not been reached.	String

Table A–64 OPMN_PROCESS Metrics

Metric	Description	Unit
type.value	The type of the process. See Table A–62 for information on process types.	
uid.value	The OPMN assigned ID for the process.	
upTime.value	The uptime for the process.	msecs

Table A–65 shows the metric table type `opmn_connect`.

Table A–65 OPMN_CONNECT Metrics

Metric	Description	Unit
desc.value	Shows the port description, if available	String
host.value	Shows the host name	String (host name)
port.value	Shows the port number	String (port number)

OPMN ONS Metrics

The Oracle Process Manager and Notification Server ONS subtree contains Oracle Notification System (ONS) information.

Table A–66 shows the metric table type `opmn_ons`.

Table A–66 OPMN_ONS Metrics

Metric	Description	Unit
notifProcessed.value	The number of notifications processed by ONS.	ops
notifProcessQueue.value	The number of notifications in the process queue.	ops
notifReceived.value	The number of notifications received by ONS.	ops
notifReceiveQueue.value	The number of notifications in the receive queue.	ops
workerThread.value	The number of worker threads.	String (threads)

Table A–67 shows the local_port metrics. The `../ons/local_port` subtree shows information about the ONS local port.

The metric table type is `opmn_connect`

Table A–67 OPMN ONS LOCAL_PORT Metrics

Metric	Description	Unit
`desc.value`	Port description	String
`host.value`	Host name	String
`port.value`	Port number	String

Table A–68 shows the remote_port metrics. The `../ons/remote_port` subtree shows information about the ONS remote port.

The metric table type is `opmn_connect`

Table A–68 OPMN ONS REMOTE_PORT Metrics

Metric	Description	Unit
`desc.value`	Port description	String
`host.value`	Host name	String
`port.value`	Port number	String

Table A–69 shows the request_port metrics. The `../ons/request_port` subtree shows information about the ONS request port.

The metric table type is `opmn_connect`

Table A–69 OPMN ONS REQUEST_PORT Metrics

Metric	Description	Unit
`desc.value`	Port description	String
`host.value`	Host name	String
`port.value`	Port number	String

Oracle Application Server Discoverer Metrics

Oracle Application Server Discoverer is deployed inside OC4J as a J2EE application. The metrics that apply to a J2EE application, Web Module, Web Context, and Servlet apply to Discoverer.

See Also: "OC4J Metrics" on page A-12

The node name subtee includes the value of the attribute ID specified as part of the process-set tag in `opmn.xml`. This subtree includes all the OPMN managed and unmanaged processes which belong to this process set.

B

Component Performance Links

This Appendix includes references to the Oracle Application Server Components that include performance information in their component level documentation.

This Appendix includes the following topics:

- Oracle Application Server Toplink Performance Information
- Oracle Application Server Portal Performance Information
- Oracle Application Server ProcessConnect Performance Information
- Oracle Application Server Forms Services Performance Information
- Oracle Application Server Discoverer Performance Information
- Oracle Application Server Wireless Performance Information
- Oracle Application Server Reports Services Performance Information

Oracle Application Server Toplink Performance Information

For information on Oracle Application Server Toplink performance tuning, refer to the chapter, "Tuning for Performance", in the *Oracle Application Server TopLink Application Developer's Guide*.

Oracle Application Server Portal Performance Information

For information on OracleAS Portal, refer to the chapter, "Tuning Performance in OracleAS Portal", in the *Oracle Application Server Portal Configuration Guide*.

Oracle Application Server ProcessConnect Performance Information

For information on Oracle Application Server ProcessConnect performance, refer the chapter, "Runtime Performance Tuning", in the *Oracle Application Server ProcessConnect User's Guide*.

Oracle Application Server Forms Services Performance Information

For information on Oracle Application Server Forms Services performance, refer to the chapters, "Tracing and Diagnostics", and "Performance Tuning Considerations", in the guide, *Oracle Application Server Forms Services Deployment Guide*.

Oracle Application Server Discoverer Performance Information

For information on Oracle Application Server Discoverer performance and scalability, refer to the chapter, "Optimizing Oracle Application Server Discoverer performance and scalability", in the *Oracle Application Server Discoverer Configuration Guide*.

Oracle Application Server Wireless Performance Information

For information on Oracle Application Server Wireless performance and scalability, refer to Chapter 13, "Optimizing Transport", in the *Oracle Application Server Wireless Administrator's Guide*.

Oracle Application Server Reports Services Performance Information

For information on Oracle Application Server Reports Services performance, refer to Part IV in the guide, *Oracle Application Server Reports Services Publishing Reports to the Web*.

Index

A

access logging, 5-15
AggreSpy
  access control, 2-8
  performance monitoring, 2-5
  URL, 2-8
  using, 2-5
Application Server Control
  module metrics, 3-6
  monitoring OC4J, 4-2
  monitoring OHS performance, 3-2
  monitoring Oracle Application Server with, 2-2
  response and load metrics, 3-5
  status metrics, 3-3

B

BC4J
  deployment configuration, 6-62
  failover mode, 6-64
  performance, 6-62
  SQL-only view object, 6-63
built-in performance metrics, 2-2

C

cache hits
  increasing rate of, 7-13
cache misses, 7-13
cache size
  calculating for Web Cache, 7-3
  maximum for Web Cache, 7-3
cacheScheme datasources option, 6-15
caching
  expires technique, 8-24
  owa_cache packages, 8-27
  system-level, 8-26
  user-level, 8-26, 8-27
  validation technique, 8-19
caching rules
  priority rankings, 7-15
  response time and, 7-15
call-timeout orion-ejb-jar.xml parameter, 6-41
capacity, 1-7
  of origin servers with Web Cache, 7-9, 7-16
  of Web Cache clusters, 7-9, 7-16
check-for-updates attribute, 6-24
cluster-config element, 6-76
CMP entity beans
  lazy-loading, 6-47
command line options, 6-4
compression
  for cached documents, 7-15
concurrency
  defined, 1-2
  limiting, 1-8
concurrent users, 5-8
connection limit
  on UNIX with Web Cache, 7-8
  on Windows with Web Cache, 7-10
  Web Cache, 7-7
connection-retry-interval datasources option, 6-18
contention, 1-5
  defined, 1-2
cookies
  JAZN session cache and, 6-69
  Web Cache and, 7-13

CPUs
insufficient, 1-5
performance and Web Cache, 7-2
cx-keepAlive attribute, 6-26
cx-max attribute, 6-25
cx-min attribute, 6-25
cx-queue attribute, 6-26

D

database monitoring, 10-1
database tuning, 10-1
data-source
stmt-cache-size attribute, 6-19
datasources
cacheScheme option, 6-15
configuring, 6-13
connection-retry-interval option, 6-18
ejb aware, 6-14
emulated, 6-14
inactivity-timeout option, 6-17
max-connect-attempts option, 6-19
max-connections option, 6-15
min-connections option, 6-16
non-emulated, 6-14
stmt-cache-size option, 6-19
wait-timeout option, 6-18
debug attribute, 6-26
dedicated.rmicontext property, 6-10
DefineColumnType property, 6-10
delay-updates-until-commit orion-ejb-jar.xml parameter, 6-43
demand rate, 1-6
deployment
application, 6-77
OC4J, 6-77
performance, 6-77
dequeue-retry-count orion-ejb-jar.xml parameter, 6-57
dequeue-retry-interval orion-ejb-jar.xml parameter, 6-57
directives
See also httpd.conf directives
distributable element
web.xml, 6-76
DMS, 9-1 to 9-20
coding tips, 9-19
conditional instrumentation, 9-20
Event sensors, 9-5
using, 9-11
getSensorWeight, 9-20
instrumentation
definition of, 9-2
using, 9-8
metrics
definition of, 9-4
dumping to files, 9-20
monitoring metrics, 9-3
naming conventions, 9-17
nouns, 9-3, 9-6
naming conventions, 9-18
using, 9-9
oracle.dms.gate property, 6-9
oracle.dms.sensors property, 6-8
oracle.jdbc.DMSstatementCachingMetrics property, 6-9
oracle.jdbc.DMSStatementMetrics property, 6-9
PhaseEvent sensors, 9-4
using, 9-10
security, 9-16
sensors, 9-3
definition of, 9-4
destroying, 9-20
resetting, 9-20
State sensors, 9-5
using, 9-12
terminology, 9-3
testing metrics, 9-15
validating metrics, 9-13
dmstool
access control, 2-10
address option, 2-11, 2-15
count option, 2-11
dump option, 2-12, 2-14
format=xml option, 2-12
interval option, 2-12
list option, 2-12
options, 2-10
table option, 2-12
using, 2-10

DNS
domain name server, 5-15
do-select-before-insert orion-ejb-jar.xml parameter, 6-43
dynamic include
vs. static include, 6-38
Dynamic Monitoring Service (DMS), See also DMS
DYNAMIC_SCHEME cacheScheme value, 6-16

E

Edge Side Includes (ESI)
memory for, 7-4
response time and, 7-14
ejb-location
datasources, 6-14
EJBs
CMP
lazy-loading attribute, 6-47
enable-passivation, 6-27
metrics, A-16
monitoring, 4-4
orion-ejb-jar.xml parameters
call-timeout, 6-41
delay-updates-until-commit, 6-43
dequeue-retry-count, 6-57
dequeue-retry-interval, 6-57
do-select-before-insert, 6-43
exclusive-write-access, 6-43
findByPrimaryKey-lazy-loading, 6-43
idletime, 6-51
isolation, 6-44
lazy-loading, 6-44
listener-threads, 6-57
locking-mode, 6-44, 6-49
max-instances, 6-41, 6-50, 6-51
max-instances-threshold, 6-51
max-tx-retries, 6-42, 6-44
memory-threshold, 6-51
min-instances, 6-42
passivate-count, 6-52
pool-cache-timeout, 6-44, 6-49, 6-50
prefetch-size, 6-45
resource-check-interval, 6-52
timeout, 6-52
transaction-timeout, 6-57
update-changed-fields-only, 6-45
validity-timeout, 6-45
performance on OC4J, 6-40
server.xml
transaction-config element, 6-22
stateful session bean passivation, 6-53
enable-passivation, 6-27
entity tag caching method, 8-20
ErrorLog
directive, 5-16
Event sensors, 9-5
exclusive-write-access orion-ejb-jar.xml parameter, 6-43
expiration-setting element, 6-29
expires caching technique, 8-24
external resource file
for static text, 6-40

F

failover
BC4J, 6-64
findByPrimaryKey-lazy-loading orion-ejb-jar.xml parameter, 6-43
finder method
lazy loading, 6-47
FIXED_RETURN_NULL_SCHEME cacheScheme value, 6-16
FIXED_WAIT_SCHEME cacheScheme value, 6-16
functional demand, 1-7

G

garbage collection
application deployment and, 6-79
OC4J applications and, 6-5, 6-6
Web Cache and, 7-6
global-thread-pool
cx-keepalive attribute, 6-26
cx-max attribute, 6-25
cx-min attribute, 6-25
cx-queue attribute, 6-26
debug attribute, 6-26

keepAlive attribute, 6-25
max attribute, 6-25
queue attribute, 6-25
global-web-application.xml
cluster-config element, 6-76
expiration-setting element, 6-29
parameters, 6-34

H

hash
defined, 1-2
parameter, 5-6
heap size
setting, 6-4
hits
cache, 7-13
HostNameLookups
directive, 5-15
HTTP connections
limiting for standalone OC4J, 6-75
HTTP Server
monitoring, 3-2
HTTP server
httpd process, 8-5
httpd
HTTP server process, 8-5
httpd.conf
directives
ErrorLog, 5-16
HostNameLookups, 5-15
KeepAlive, 3-15, 5-13
KeepAliveTimeout, 5-13, 5-14
ListenBacklog, 5-12
LogLevel, 5-16
MaxClients, 5-12, 5-13
MaxKeepAliveRequests, 5-13, 5-14
MaxRequestsPerChild, 5-12
MaxSpareServers, 5-12, 8-14
MinSpareServers, 5-12, 8-14
StartServers, 5-13, 8-14
ThreadsPerChild, 5-14
Timeout, 5-13
port numbers, 4-9

I

idletime orion-ejb-jar.xml parameter, 6-51
inactivity-timeout datasources option, 6-17
include directive use with JSPs, 6-38
incoming connections
Web Cache, 7-7
isolation orion-ejb-jar.xml parameter, 6-44

J

J2EE
applications monitoring, 4-4
deployment, 6-77
guidelines for performance, 6-1
identifying data sources, 6-14
improving performance, 6-1
metrics, A-12
JAAS performance, 6-66
Java options
changing for OC4J, 6-11
-client, 6-6
-server, 6-6
-Xms, 6-4
-Xmx, 6-4
-Xss, 6-7
-XX+AggressiveHeap, 6-6
-XXMaxPermSize, 6-7
java.naming.factory.initial property, 6-73
java.naming.provider.url property, 6-73
java.naming.security.credentials property, 6-74
java.naming.security.principal property, 6-74
java.security.auth.policy property, 6-68
javax.sql.DataSource, 6-13
jaxn.xml configuration file, 6-68
JAZN performance, 6-66
JDBC
DefineColumnType option, 6-10
metrics, 6-9, A-8
oracle.jdbc.DMSStatementCachingMetrics property, 6-9
oracle.jdbc.DMSStatementMetrics property, 6-9
prefetch-size parameter, 6-20
statement cache size attribute, 6-19
stmt-cache-size, 6-19

JServ
metrics, A-43
JSPs
configuration
main_mode, 6-34
dynamic include, 6-39
include directives, 6-38
justrun main_mode parameter, 6-34
metrics, A-14
monitoring, 4-4
page buffer, 6-38
page sessions, 6-36
performance, 6-33
recompile main_mode parameter, 6-34
reload main_mode parameter, 6-34
runtime include, 6-39
static include, 6-38
translate-time includes, 6-38
justrun main_mode parameter, 6-34
JVM
command line options, 6-4
metrics, A-7
setting heap size, 6-4

K

KeepAlive httpd.conf directive, 3-15, 5-13, 5-14, 8-39
Web Cache and, 7-12
keepAlive server.xml attribute, 6-25
Keep-Alive timeout
Web Cache setting, 7-11
KeepAliveTimeout httpd.conf directive, 5-13, 5-14
Web Cache and, 7-12

L

latency
defined, 1-2
first-request, 6-29
lazy-loading attribute, 6-47
lazy-loading orion-ejb-jar.xml parameter, 6-44
ListenBacklog httpd.conf directive, 5-12
listener-threads orion-ejb-jar.xml parameter, 6-57
load variances, 1-9
load-on-startup web.xml parameter, 6-29
locking-mode orion-ejb-jar.xml parameter, 6-44, 6-49
locking-mode values
optimistic, 6-46
pessimistic, 6-46
read-only, 6-46
logging
access, 5-15
error, 5-16
performance and, 5-15
performance implications of, 5-15
LogLevel directive, 5-16
logresolve utility, 5-15

M

main_mode parameter, 6-34
max attribute, 6-25
MaxClients httpd.conf directive, 5-12, 5-13
Web Cache and, 7-12, 7-15
max-connect-attempts datasources option, 6-19
max-connections datasources option, 6-15
max-connections-queue-timeout attribute, 6-75
max-http-connections element, 6-75
maximum cache size
configuring for Web Cache, 7-3
maximum network connections
Web Cache and, 7-7
max-instances orion-ejb-jar.xml parameter, 6-41, 6-50, 6-51
max-instances-threshold orion-ejb-jar.xml parameter, 6-51
MaxKeepAliveRequests httpd.conf directive, 5-13, 5-14
Web Cache and, 7-12
MaxRequestsPerChild httpd.conf directive, 5-12
MaxSpareServers httpd.conf directive, 5-12
max-tx-retries orion-ejb-jar.xml parameter, 6-42, 6-44
memory
calculating for Web Cache, 7-3
configuring for Web Cache, 7-3
ESI and Web Cache, 7-4
JVM heap size, 6-4

memory-threshold orion-ejb-jar.xml parameter, 6-51
metric table types
JDBC_Connection, A-9, A-10
JDBC_DataSource, A-8
JDBC_Driver, A-8
JDBC_Statement, A-10
JMSBrowserStats, A-24
JMSConnectionStats, A-22
JMSDestinationStats, A-26
JMSDurableSubscriptionStats, A-25
JMSMessageConsumerStats, A-25
JMSPersistenceStats, A-28
JMSProducerStats, A-24
JMSRequestHandlerStats, A-22
JMSSessionStats, A-23
JMSStats, A-20
JMSStoreStats, A-27
JMSTempoaryDestinationStats, A-26
jserv_jsp, A-46
jserv_server, A-43
jserv_servlet, A-45
jserv_zone, A-44
JVM, A-7
mod_oc4J_destination_metrics, A-6
mod_oc4J_mount_pt_metrics, A-5
mod_oc4j_request_failure_causes, A-4
modplsql_Cache, A-31
modplsql_DatabaseConnectionPool, A-32, A-33
modplsql_HTTPResponseCodes, A-3, A-30
modplsql_LastNSQLError, A-32
modplsql_PageEngine, A-35
modplsql_PageEngine_ResponseCodes, A-37
modplsql_SQLErrorGroup, A-31
oc4j_context, A-13
oc4j_ejb_entity_bean, A-16
oc4j_ejb_method, A-16
oc4j_jsp(threadsafe=false), A-15
oc4j_jsp(threadsafe=true), A-15
oc4j_jspExec, A-14
oc4J_opmn, A-18
oc4j_servlet, A-14
oc4j_task, A-28
oc4j_web_module, A-12
ohs_module, A-4
ohs_server, A-2
opmn_connect, A-52, A-53
opmn_host_statistics, A-49
opmn_ias_instance, A-50
opmn_ons, A-52
opmn_pm, A-49
opmn_process, A-50
opmn_process_set, A-50
opmn_process_type, A-50
metric tables, 2-6
metrics
acknowledgeMode.value, A-23
activeConnections.maxValue, A-44
activeConnections.value, A-44
activeInstances.value, A-15
activeSessions.value, A-44
activeSTMInstances.maxValue, A-46, A-48
activeSTMInstances.value, A-46, A-48
activeThreadGroups.maxValue, A-7
activeThreadGroups.minValue, A-7
activeThreadGroups.value, A-7
activeThreads.maxValue, A-7
activeThreads.minValue, A-7
activeThreads.value, A-7
address.value, A-21, A-22, A-23
availableInstances.value, A-15
bean-type.value, A-16
cacheEnabled.value, A-36
CacheFreeSize.value, A-8
CacheGetConnection.avg, A-8
CacheGetConnection.completed, A-8
CacheGetConnection.maxTime, A-9
CacheGetConnection.minTime, A-9
CacheGetConnection.time, A-9
CacheHit.count, A-9
cacheHits.value, A-38, A-39, A-40, A-41, A-42
CacheMiss.count, A-9
cachePageHits.value, A-36
cachePageRequests.value, A-36
CacheSize.value, A-9
cacheStatus.value, A-31
checkReload.active, A-44
checkReload.avg, A-44
checkReload.completed, A-44
checkReload.maxTime, A-44

checkReload.minTime, A-44
checkReload.time, A-44
client.active, A-17
client.avg, A-17
client.completed, A-17
clientID.value, A-23, A-26
client.maxTime, A-17
client.minTime, A-17
client.time, A-17
connection.active, A-2
connection.avg, A-2
ConnectionCloseCount.count, A-8
ConnectionCreate.active, A-8
ConnectionCreate.avg, A-8
ConnectionCreate.completed, A-8
ConnectionCreate.maxTime, A-8
ConnectionCreate.minTime, A-8
ConnectionCreate.time, A-8
connectionID.value, A-22, A-26
connection.maxTime, A-2
connection.minTime, A-2
ConnectionOpenCount.count, A-8
connections.count, A-21
connection.time, A-2
connFetch.active, A-33, A-34
connFetch.avg, A-33, A-34
connFetch.completed, A-33, A-34, A-38, A-39, A-40, A-41, A-42, A-43
connFetch.maxTime, A-33, A-34
connFetch.minTime, A-33, A-34
connFetch.time, A-33, A-34
cpuIdle.value, A-49
cpuTime.value, A-51
CreateNewStatement.avg, A-9, A-10
CreateNewStatement.completed, A-9, A-10
CreateNewStatement.maxTime, A-9, A-10
CreateNewStatement.minTime, A-9, A-10
CreateNewStatement.time, A-9, A-10
createSession.active, A-46, A-47
createSession.avg, A-46, A-47
createSession.completed, A-46, A-48
createSession.maxTime, A-46, A-48
createSession.minTime, A-46, A-48
createSession.time, A-46, A-48
CreateStatement.avg, A-9, A-10
CreateStatement.completed, A-9, A-10
CreateStatement.maxTime, A-9, A-10
CreateStatement.minTime, A-9, A-10
CreateStatement.time, A-9, A-10
default_application_log.value, A-18
deliveryMode.value, A-24
desc.value, A-53
Destination.value, A-5
destination.value, A-24, A-25, A-26, A-27, A-28
disableMessageID.value, A-24
disableMessageTimestamp.value, A-24
domain.value, A-23, A-24, A-25, A-26
EJB, A-16
ejbPostCreate.active, A-17
ejbPostCreate.avg, A-17
ejbPostCreate.completed, A-17
ejbPostCreate.maxTime, A-17
ejbPostCreate.minTime, A-17
ejbPostCreate.time, A-17
error.count, A-32
errorDate.value, A-32
errorRequest.value, A-32
errorText.value, A-32
ErrReq.count, A-5, A-6
ErrReqNonSess.count, A-5, A-6
ErrReqSess.count, A-5, A-6
exceptionListener.value, A-23
exclusive-write-access.value, A-16
Execute.time, A-11
executeTime.active, A-38, A-39, A-40, A-41, A-42, A-43
executeTime.avg, A-38, A-39, A-40, A-41, A-42
executeTime.maxTime, A-38, A-39, A-40, A-41, A-42
executeTime.minTime, A-38, A-39, A-40, A-41, A-42
executeTime.time, A-38, A-39, A-40, A-41, A-42, A-43
Failover.count, A-5, A-6
Fetch.time, A-11
freeMemory.maxValue, A-7
freeMemory.minValue, A-7
freeMemory.value, A-7
freePhysicalMem.value, A-49
handle.active, A-2, A-4

handle.avg, A-2, A-4
handle.completed, A-2, A-4
handle.maxTime, A-2, A-4
handle.minTime, A-2, A-4
handle.time, A-2, A-4
heapSize.value, A-51
hits.count, A-31, A-33, A-34
holePageCount.value, A-28
host.value, A-21, A-22, A-23, A-44, A-53
httpTimeout.value, A-38
httpUnresolvedRedirect.value, A-38
httpXXX.value, A-38, A-39, A-40, A-41, A-42
ias_cluster.value, A-18
ias_instance.value, A-18
iasCluster.value, A-50, A-51
iasInstance.value, A-51
idlePeriod.completed, A-44
idlePeriod.maxTime, A-44
idlePeriod.minTime, A-44
idlePeriod.time, A-44
IncorrectReqInit.count, A-5
indexInSet.value, A-51
interval.value, A-28
isActive.value, A-26
isLocal.value, A-23
isolation.value, A-16
isOpen.value, A-28
isXA.value, A-23
J2EE, A-12
jms_log.value, A-18
jobWorkerQueue.value, A-49
JServ, A-43
JSP, A-14
JVM, A-7
JVMCnt.value, A-6
lastErrorDate.value, A-32
lastErrorRequest.value, A-32
lastErrorText.value, A-32
lastResponseCode.value, A-38, A-39, A-40, A-41, A-42
lastResponseDate.value, A-38, A-39, A-40, A-41, A-42
lastUsed.value, A-28
loadFailed.count, A-45
loadServlet.avg, A-46, A-47
loadServletClasses. active, A-47
loadServletClasses.active, A-46
loadServletClasses.avg, A-46, A-47
loadServletClasses.completed, A-46, A-47
loadServletClasses.maxTime, A-46, A-47
loadServletClasses.minTime, A-46, A-47
loadServletClasses.time, A-46, A-47
loadServlet.completed, A-46, A-47
loadServlet.maxTime, A-46, A-47
loadServlet.minTime, A-46, A-47
loadServlet.time, A-46, A-47
locations.value, A-26
LogicalConnection.value, A-9, A-10
lReq.count, A-49
maxBacklog.value, A-44
maxConnections.value, A-44
maxSTMInstances.value, A-46, A-48
memoryUsed.value, A-51
messageCount.value, A-27
messageDequeued.count, A-27
messageDiscarded.count, A-27
messageEnqueued.count, A-27
messageExpired.count, A-27
messageListener.value, A-25
messagePagedIn.count, A-27
messagePagedOut.count, A-27
messageRecovered.count, A-27
moduleId.value, A-50
Name.value, A-5, A-6
name.value, A-25, A-26
newMisses.count, A-31, A-33, A-34
noLocal.value, A-25, A-26
NonSessFailover.count, A-5, A-7
notifProcessed.value, A-52
notifProcessQueue.value, A-52
notifReceived.value, A-52
numMods.value, A-4
numProcConf.value, A-50
numProcessors.value, A-49
oc4j_instance.value, A-18
oc4j_island.value, A-18
oc4j.jms.computeMsgsize.value, A-21
oc4j.jms.debug.value, A-21
oc4j.jms.doGc.value, A-21
oc4j.jms.expirationInterval.value, A-21

oc4j.jms.forceRecovery.value, A-21
oc4j.jms.intraSession.value, A-21
oc4j.jms.j2ee14.value, A-21
oc4j.jms.lazySync.value, A-21
oc4j.jms.listenerAttempts.value, A-21
oc4j.jms.maxOpenFiles.value, A-21
oc4j.jms.messagePoll.value, A-21
oc4j.jms.noDms.value, A-21
oc4j.jms.pagingThreshold.value, A-21
oc4j.jms.saveAllExpired.value, A-21
oc4j.jms.serverPoll.value, A-21
oc4j.jms.socketBufsize.value, A-21
oc4j.jms.usePersistence.value, A-21
oc4j.jms.useSockets.value, A-21
oc4j.jms.useUUID.value, A-22
Oc4jUnavailable.count, A-5
offline.value, A-40, A-42
opmn_group.value, A-18
opmn_sequence.value, A-18
Oracle Application Server performance, A-1
pageElapsedTimeAvg.count, A-36
pageElapsedTimeAvg.value, A-36
pageElapsedTime.count, A-36
pageElapsedTime.maxValue, A-36
pageElapsedTime.minValue, A-36
pageElapsedTime.value, A-36
pageMetadataFetchTimeAvg.count, A-37
pageMetadataFetchTimeAvg.value, A-36
pageMetadataFetchTime.count, A-37
pageMetadataFetchTime.maxValue, A-37
pageMetadataFetchTime.minValue, A-37
pageMetadataFetchTime.value, A-37
pageMetadataWaitTimeAvg.count, A-36
pageMetadataWaitTimeAvg.value, A-36
pageMetadataWaitTime.count, A-36
pageMetadataWaitTime.maxValue, A-36
pageMetadataWaitTime.minValue, A-36
pageMetadataWaitTime.value, A-36
pageRequests.value, A-36
parseRequest.active, A-12
parseRequest.avg, A-12
parseRequest.completed, A-12
parseRequest.maxTime, A-12
parseRequest.minTime, A-12
parseRequest.time, A-12
pendingMessageCount.value, A-27
persistenceFile.value, A-28
persistence-type.value, A-16
pid.value, A-51
portal, A-29
port.value, A-22, A-23, A-44, A-53
priority.value, A-24
privateMemory.value, A-51
procDeath.count, A-49
procDeathReplace.count, A-49
processRequest.active, A-12, A-15, A-45, A-47
processRequest.avg, A-12, A-15, A-45, A-47
processRequest.completed, A-12, A-15, A-45, A-47
processRequest.maxTime, A-12, A-15, A-45, A-47
processRequest.minTime, A-12, A-15, A-45, A-47
processRequest.time, A-12, A-15, A-45, A-47
queueLengthAvg.count, A-37
queueLengthAvg.value, A-37
queueLength.count, A-37
queueLength.maxValue, A-37
queueLength.minValue, A-37
queueLength.value, A-37
queueStayAvg.count, A-37
queueStayAvg.value, A-37
queueStay.count, A-37
queueStay.maxValue, A-37
queueStay.minValue, A-37
queueStay.value, A-37
queueTimeout.value, A-37
readRequest.active, A-44
readRequest.avg, A-44
readRequest.completed, A-44
readRequest.maxTime, A-44
readRequest.minTime, A-44
readRequest.time, A-44
readSession.count, A-45
reqFail.count, A-49, A-50
reqPartialSucc.count, A-49, A-50
reqSucc.count, A-49, A-50
request.active, A-2
request.avg, A-2
request.completed, A-2

requestHandlers.count, A-22
request.maxTime, A-2
request.minTime, A-2
requests.count, A-31
request.time, A-2
resolveContext.active, A-13
resolveContext.avg, A-13
resolveContext.completed, A-13
resolveContext.maxTime, A-13
resolveContext.minTime, A-13
resolveContext.time, A-13
resolveServlet.avg, A-13
resolveServlet.completed, A-13
resolveServlet.maxTime, A-13
resolveServlet.minTime, A-13
resolveServlet.time, A-13
restartOnDeath.value, A-50
rmi_log.value, A-18
rReq.count, A-49
run().active, A-28
run().avg, A-28
run().completed, A-28
run().maxActive, A-28
run().maxTime, A-28
run().minTime, A-28
run().time, A-28
selector.value, A-25, A-26
server_log.value, A-18
service.active, A-14, A-15
service.avg, A-14, A-15
service.completed, A-13, A-14, A-15
service.maxTime, A-14, A-15
service.minTime, A-14, A-15
serviceRequest.active, A-45, A-47
serviceRequest.avg, A-45, A-47
serviceRequest.completed, A-45, A-47
serviceRequest.maxTime, A-45, A-47
serviceRequest.minTime, A-45, A-47
serviceRequest.time, A-45, A-47
service.time, A-13, A-14, A-15
SessFailover.count, A-6, A-7
sessionActivation.avg, A-13
sessionActivation.completed, A-13
sessionActivation.maxTime, A-13
sessionActivation.minTime, A-13
sessionActivation.time, A-13
sessionListener.value, A-23
session-type.value, A-16
sharedMemory.value, A-51
SQLText.value, A-11
staleMisses.count, A-31, A-33, A-34
startTime.value, A-22, A-23, A-24, A-25
opmn_process, A-51
status.value, A-51
storeSize.value, A-27
SucReq.count, A-6, A-7
SucReqNonSess.count, A-6, A-7
SucReqSess.count, A-6, A-7
taskManagerInterval.value, A-22
timestamp.value, A-49
timeToLive.value, A-24
totalMemory.maxValue, A-8
totalMemory.minValue, A-8
totalMemory.value, A-8
totalPhysicalMem.value, A-49
transacted.value, A-23
transaction-type.value, A-16
trans-attribute.value, A-17
txid.value, A-24
type.value, A-52
uid.value, A-52
UnableToHandleReq.count, A-5
upTime.value, A-7, A-52
usedPageCount.value, A-28
user.value, A-23
workerThread.value, A-49, A-52
wrapper.active, A-17
wrapper.avg, A-18
wrapper.completed, A-18
wrapper.maxTime, A-18
wrapper.minTime, A-18
wrapper.time, A-18
writeSession.count, A-45
xid.value, A-24
min-connections datasources option, 6-16
min-instances orion-ejb-jar.xml parameter, 6-42
MinSpareServers httpd.conf directive, 5-12
misses
cache, 7-13
mod_expires, 8-39

modplsql_Cache
metric table type, A-31
modplsql_DatabaseConnectionPool
metric table type, A-32, A-33
modplsql_HTTPResponseCodes
metric table type, A-3, A-30
modplsql_LastNSQLError
metric table type, A-32
modplsql_PageEngine
metric table type, A-35
modplsql_PageEngine_ResponseCodes
metric table type, A-37
modplsql_SQLErrorGroup
metric table type, A-31
monitoring
EJBs, 4-4
JSPs, 4-4
OC4J
J2EE applications monitoring, 4-4
Oracle HTTP Server, 3-11
performance statistics, 2-2
servlets, 4-4

N

naming conventions
DMS, 9-17
network
bandwidth and Web Cache, 7-7
connections and Web Cache, 7-7
on UNIX, 7-8
on Windows, 7-10
network parameters
setting for Web Cache, 7-11
tuning, 5-2
network performance, 5-9
no-reverseping-failed-ping-limit parameter, 6-79
nouns
creating, 9-9
DMS, 9-6
naming conventions, 9-18
type, 9-6

O

OC4J
applications monitoring, 4-4
Instance monitoring, 4-2
monitoring performance statistics, 2-2
process monitoring, 4-2
properties
dedicated.rmicontext, 6-10, 6-74
DefineColumnType, 6-10
oracle.dms.sensors, 6-8
oracle.jdbc.DMSStatementCachingMetrics, 6-9
oracle.jdbc.DMSStatementMetrics, 6-9
oc4j_context
metric table type, A-13
oc4j_ejb_entity_bean
metric table type, A-16
oc4j_ejb_method
metric table type, A-16
oc4j_jspExec
metric table type, A-14
oc4j_servlet
metric table type, A-14
oc4j_web_module
metric table type, A-12
opmn.xml parameter
no-reverseping-failed-ping-limit, 6-79
reverseping-failed-ping-limit, 6-79
optimistic locking-mode value, 6-46
Oracle Application Server Web Cache. See Web Cache
Oracle Business Components for Java. See BC4J
Oracle HTTP Server
configuring with directives, 5-11
monitoring, 3-11
oracle.dms.gate property, 6-9
oracle.dms.sensors property, 6-8
oracle.jdbc.DMSStatementCachingMetrics property, 6-9
oracle.jdbc.DMSStatementMetrics property, 6-9
oracle.security.jazn.config property, 6-68
origin server timeout
Web Cache setting, 7-11
owa_cache package, 8-27

P

page buffers with JSPs, 6-38
parameters
hash, 5-6
KeepAlive, 3-14, 5-13, 5-14
KeepAliveTimeout, 5-13
ListenBacklog, 5-12
MaxClients, 5-12, 5-13, 8-8
MaxKeepAliveRequests, 5-13
MaxRequestsPerChild, 5-12, 8-7
MaxSpareServers, 5-12, 8-8
MinSpareServers, 5-12, 8-8
PlsqlIdleSessionCleanupInterval, 8-7
PlsqlMaxRequestsPerSession, 8-7
StartServers, 5-13
TCP, 5-2
tcp_conn_hash_size, 5-2, 5-6, 5-7
tcp_conn_req_max_q, 5-2, 5-8
tcp_conn_req_max_q0, 5-2, 5-8
tcp_recv_hiwat, 5-2
tcp_slow_start_initial, 5-2
tcp_time_wait_interval, 5-2, 5-6
tcp_xmit_hiwat, 5-2
ThreadsPerChild, 5-14
Timeout, 5-13
partial page caching
Web Cache and, 7-14
passivate-count orion-ejb-jar.xml parameter, 6-52
passivation
sfsb-config enable-passivation attribute, 6-27
stateful session bean, 6-53
performance
application deployment, 6-77
goals, 1-7
monitoring
native operating system, 2-4
network monitoring tools, 2-4
tuning
expires caching, 8-24
mod_expires, 8-39
system-level caching, 8-26
validation caching, 8-20
Web Cache and, 7-1
Web Cache and CPUs, 7-2
persistent connections
KeepAlive directive, 5-14
pessimistic locking-mode value, 6-46
PhaseEvent sensors, 9-4
PL/SQL web toolkit functions, 8-27
pool-cache-timeout orion-ejb-jar.xml parameter, 6-44, 6-49, 6-50
portal
metrics, A-29
prefetch-size orion-ejb-jar.xml parameter, 6-20, 6-45
processes used
Web Cache, 7-2

Q

queue attribute, 6-25

R

read-only locking-mode value, 6-46
recompile main_mode parameter, 6-34
redirection
Web Cache and, 7-14
reload main_mode parameter, 6-34
resource-check-interval orion-ejb-jar.xml parameter, 6-52
response time, 1-5
defined, 1-2
goal, 1-8
improving, 1-3
optimizing for Web Cache, 7-15
peak load, 1-9
reverseping-failed-ping-limit parameter, 6-79

S

scalability
defined, 1-2
Server Properties, 5-9
server.xml
check-for-updates attribute, 6-24
global-thread-pool element, 6-24
max-http-connections, 6-75
parameters, 6-22

sfsb-config element, 6-27
taskmanager-granularity element, 6-23
service time, 1-3, 1-5
defined, 1-2
servlets
loading on startup, 6-29
monitoring, 4-4
unused sessions, 6-31
sessions
use with JSPs, 6-36
session-timeout attribute, 6-30, 6-69
sfsb-config
element, 6-27
passivation control, 6-53
sock-backlog attribute, 6-75
StartServers httpd.conf directive, 5-13
State sensors, 9-5
static include
vs. dynamic include, 6-38
static text
external resource file, 6-40
statistics
cache size for Web Cache, 7-6
memory for Web Cache, 7-6
stmt-cache-size attribute, 6-19
system-level caching, 8-26

T

taskmanager-granularity, 6-23
TCP
parameters, 5-2
setting parameters, 5-6
think time
defined, 1-2
ThreadsPerChild directive, 5-14
throughput
defined, 1-2
demand limiter and, 1-7
increasing, 1-5
Timeout httpd.conf directive, 5-13
timeout orion-ejb-jar.xml parameter, 6-52
time-wait settings
Web Cache and, 7-12
transaction-config server.xml parameter, 6-22
transaction-timeout orion-ejb-jar.xml parameter, 6-57
tuning
expires caching technique, 8-24
system-level caching, 8-26
validation caching, 8-20

U

unit consumption, 1-7
unused sessions
servlets, 6-31
update-changed-fields-only orion-ejb-jar.xml parameter, 6-45
URL parameters
Web Cache and, 7-13
user-level caching, 8-27

V

validation caching
for mod_plsql, 8-20
technique, 8-19
validity-timeout orion-ejb-jar.xml parameter, 6-45

W

wait time
contention and, 1-5
defined, 1-3
parallel processing and, 1-4
wait-timeout datasources option, 6-18
Web Cache
cache hit rates, 7-13
calculating memory and cache size, 7-3
configuring memory and cache size, 7-3
Edge Side Includes (ESI), 7-4
garbage collection, 7-6
guidelines for performance, 7-1
improving performance, 7-1
network bandwidth, 7-7
network connections, 7-7
on UNIX, 7-8
on Windows, 7-10
network parameters, 7-11

partial page caching, 7-14
performance and CPUs, 7-2
processes used, 7-2
response time, 7-15
statistics for memory and cache size, 7-6
Web Services
performance issues, 6-61
web toolkit, 8-27
web.xml
distributable element, 6-76
dmsLogging parameter, 6-9
load-on-startup parameter, 6-29
session-config element, 6-30, 6-69

X

xml format output for dmstool, 2-12