Oracle9*i* Application Server

Performance Guide

Release 2 (9.0.2) for UNIX

April 2002

Part No. A97380-01

ORACLE

Oracle9*i* Application Server Performance Guide, Release 2 (9.0.2) for UNIX

Part No. A97380-01

Contributors: Thomas Van Raalte, 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

2 Monitoring Oracle9*i*AS

3 Monitoring Oracle HTTP Server

4 Monitoring OC4J

5 Optimizing Oracle HTTP Server

6 Optimizing J2EE Applications In OC4J

A Oracle9*i*AS Performance Metrics

Index

Send Us Your Comments

Oracle9*i* Application Server Performance Guide, Release 2 (9.0.2) for UNIX

Part No. A97380-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. 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?

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

- Electronic mail: iasdocs_us@oracle.com
- FAX: 650-506-7407 Attn: Oracle9*i* Application Server Documentation Manager
- Postal service:
  Oracle Corporation
  Oracle9*i* Application Server Documentation
  500 Oracle Parkway, M/S 2op3
  Redwood Shores, CA 94065
  USA

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

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 Oracle9*i* Application Server environment.

This preface contains these topics:

Intended Audience

Oracle9i Application Server Performance Guide is intended for Internet application developers, Oracle9*i* 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 Oracle9*i*AS performance and tuning concepts.

Chapter 2, "Monitoring Oracle9iAS"

This chapter introduces the available performance monitoring tools, including Oracle Enterprise Manager and the built-in Oracle9*i*AS performance monitoring tools.

Chapter 3, "Monitoring Oracle HTTP Server"

This chapter discusses monitoring the Oracle HTTP Server using Oracle Enterprise Manager and the built-in performance tools available with Oracle9*i*AS.

Chapter 4, "Monitoring OC4J"

This chapter discusses monitoring Oracle9*i*AS Containers for J2EE(OC4J) using Oracle Enterprise Manager and the built-in performance tools available with Oracle9*i*AS.

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 Oracle9*i*AS Containers for J2EE.

Chapter 7, "Optimizing Web Cache"

This chapter discusses optimizing Web Cache.

Chapter 8, "Optimizing PL/SQL Performance"

This chapter discusses optimizing code using `mod_plsql`.

Appendix A, "Oracle9iAS Performance Metrics"

This chapter discusses the statistics and metrics used to monitor and analyze the performance of Oracle9*i*AS components.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Application Server Concepts Guide*
- *Oracle9i Application Server Administrator's Guide*
- *Oracle HTTP Server Administration Guide*
- *Oracle9iAS Containers for J2EE User's Guide*
- *Oracle9i Application Server Security Guide*
- *Oracle9iAS Web Cache Administration and Deployment Guide*
- *Oracle9iAS Containers for J2EE Enterprise JavaBeans Developer's Guide and Reference*
- *Oracle9iAS Containers for J2EE Servlet Developer's Guide*
- *Oracle9iAS Containers for J2EE JSP Tag Libraries and Utilities Reference*
- *Oracle9i Database Performance Tuning Guide and Reference*
- *Oracle9i Application Server 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 Oracle9*i*AS Portal performance at:

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

  From the Oracle Technology Network main page:

  - Choose the Product link
  - Choose Oracle9*i*AS Portal under Oracle9*i* 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 Oracle 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:\winnt"\"system32` is the same as `C:\WINNT\SYSTEM32`
`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 Oracle8*i* 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:\orant` for Windows NT ■ `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 Oracle9*i* 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 Oracle9*i* Application Server performance and tuning concepts.

This chapter contains the following sections:

- Introduction to Oracle9iAS Performance
- What Is Performance Tuning?
- Performance Targets
- Performance Methodology

> **See Also:** *Oracle9i Application Server Concepts Guide* for a discussion of Oracle9*i* Application Server concepts

Introduction to Oracle9*i*AS Performance

To maximize Oracle9*i* 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 Oracle9*i*AS components, such as Oracle HTTP Server and Oracle9*i*AS 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 1-8 for a discussion on performance requirements and determining what parts of the system to tune.

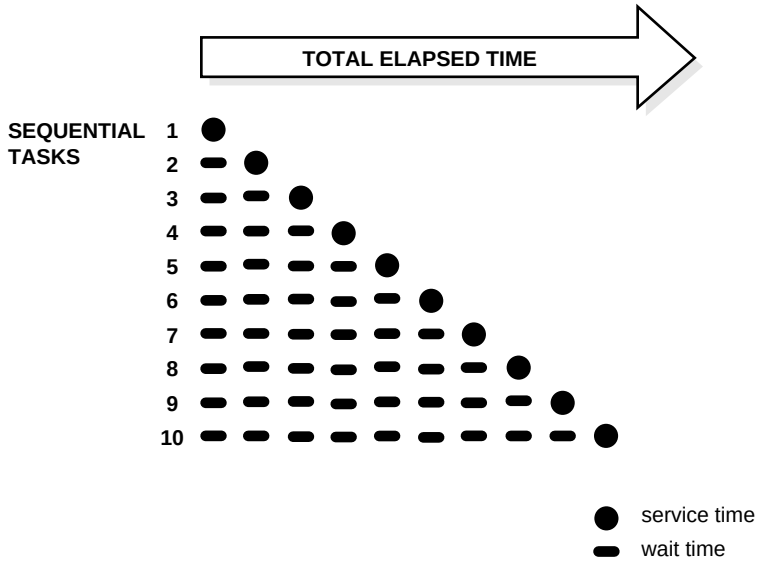
Response Time

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` parameter 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

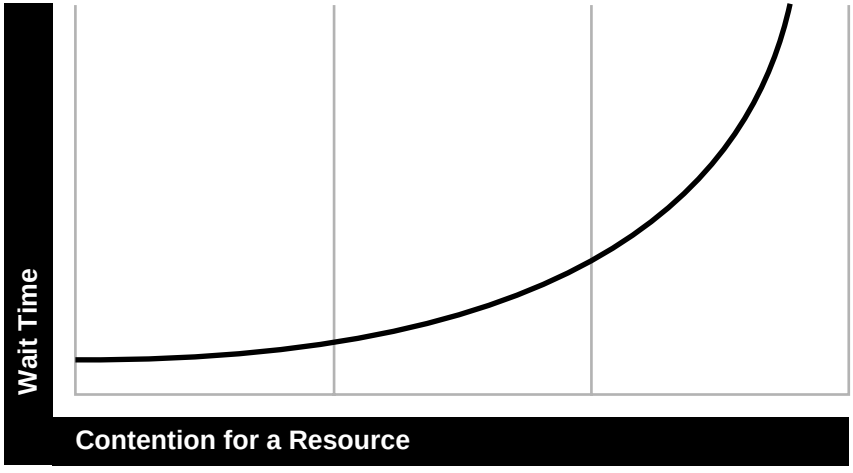
System **throughput** is the amount of work accomplished in a given amount of time. You can increase **throughput** by:

- 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



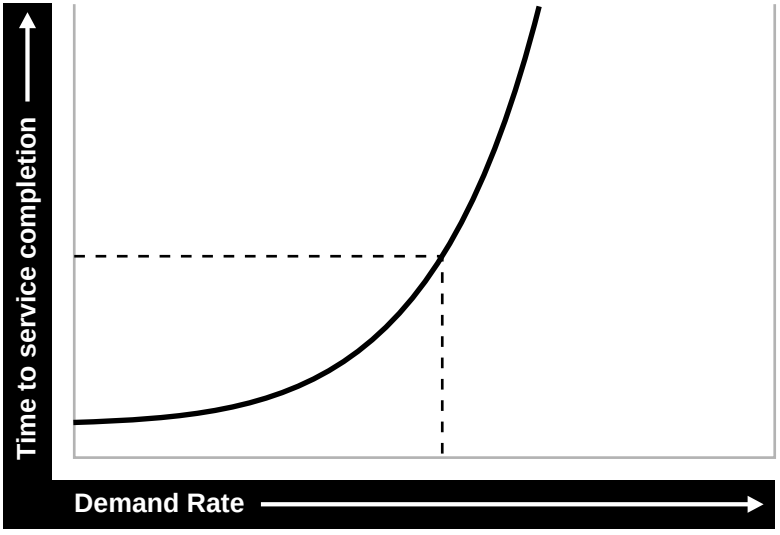
Critical Resources

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



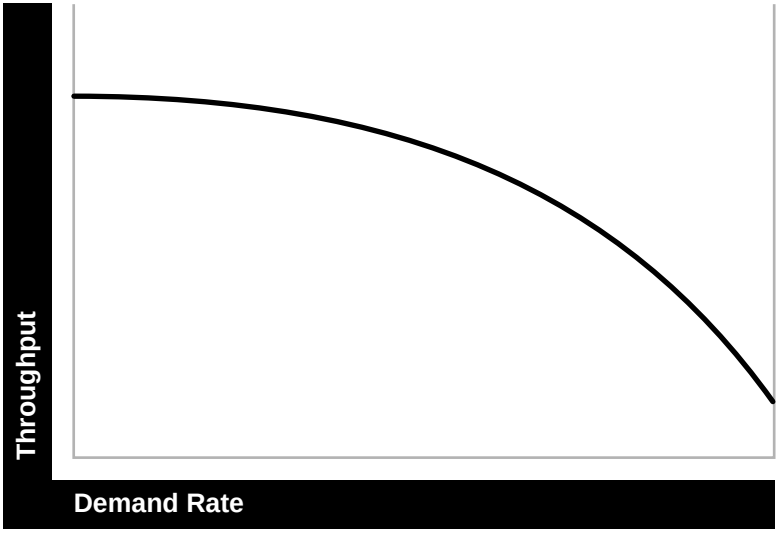
To manage this situation, you have two options:

- 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.

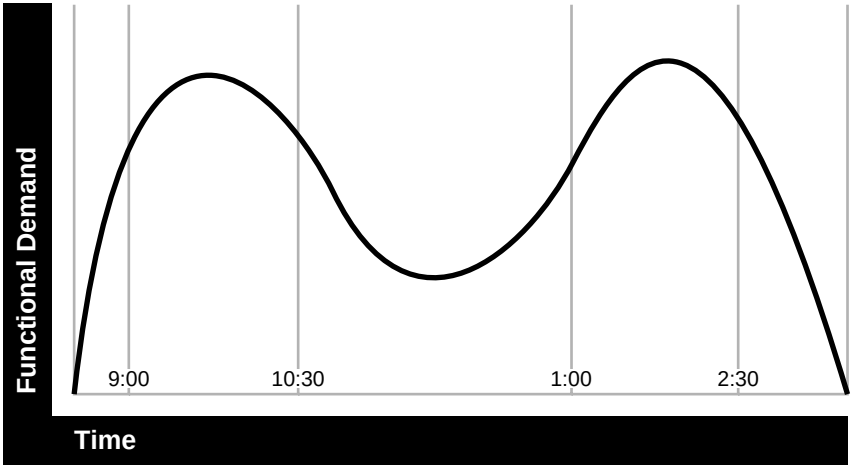
Ongoing performance monitoring enables you to maintain a well tuned system. Keeping a history of the application's performance over time enables you to make useful comparisons. With data about actual resource consumption for a range of loads, you can conduct objective **scalability** studies and from 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 Oracle9*i*AS

This chapter discusses how to monitor the performance of Oracle9*i*AS and its components. Obtaining performance data can assist you in tuning Oracle9*i*AS or in tuning and debugging applications with performance problems.

This chapter contains the following topics:

Overview of Monitoring Oracle9*i*AS

This section describes how to use the available tools for performance monitoring. You can monitor Oracle9*i*AS and its components using one or more of the following:

- Oracle Enterprise Manager
- Oracle9iAS Built-in Performance Metrics
- Native Operating System Performance Commands
- Network Performance Monitoring Tools

Oracle Enterprise Manager

Oracle Enterprise Manager allows you to monitor Oracle9*i*AS and its components. Oracle Enterprise Manager shows performance metrics for Oracle9*i*AS components, including:

- Oracle HTTP Server (OHS)
- Oracle9*i*AS Containers for J2EE (OC4J) and Applications running under OC4J
- Oracle9*i*AS Web Cache
- Oracle9*i*AS Portal

See Also:

- Chapter 3, "Monitoring Oracle HTTP Server"
- Chapter 4, "Monitoring OC4J"
- *Oracle9i Application Server Administrator's Guide*

Oracle9*i*AS Built-in Performance Metrics

Oracle9*i*AS automatically measures runtime performance and collects metrics for the Oracle HTTP Server including child servers and Oracle9*i*AS Containers for J2EE (OC4J) servers. The Oracle9*i*AS performance metrics are measured automatically and continuously using performance instrumentation inserted into the implementations of Oracle9*i*AS components. The performance metrics are automatically enabled; you do not need to set options or perform any extra configuration to collect the performance metrics.

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.

You can use the performance metrics while troubleshooting Oracle9*i*AS 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.

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 Oracle9*i*AS 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 Oracle9*i*AS Built-in Performance Metrics

You can monitor Oracle9*i*AS performance using Oracle Enterprise Manager or by viewing the Oracle9*i*AS built-in performance metrics.

This section describes how to view the Oracle9*i*AS built-in performance metrics using the `AggreSpy` servlet or using the `dmstool` command. Table 2–1 briefly describes these methods for viewing the built-in performance metrics.

Table 2–1 Oracle9iAS Built-in Monitoring Commands

Command	Description
`AggreSpy`	`AggreSpy` is a pre-packaged servlet that reports performance metrics. `AggreSpy` reports performance data for an Oracle9*i*AS instance. You can only run `AggreSpy` when an OC4J instance is configured to support it. By default, the OC4J home instance 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 every *t* seconds. This command also allows you to report all the built-in performance metrics on your site. `dmstool` is located in the `$ORACLE_HOME/bin` directory.

See Also:

- Appendix A, "Oracle9iAS Performance Metrics"

Viewing Performance Metrics Using AggreSpy

The `AggreSpy` servlet displays Oracle HTTP Server and OC4J performance metrics. `AggreSpy` provides output using easy to read HTML tables and includes OC4J metrics from multiple OC4J instances when multiple OC4Js are running.

The tables containing built-in metrics in Oracle9*i*AS are identified by a name, such as `ohs_server` for the Oracle HTTP Server metrics. In the text in this guide we refer to the built-in performance metric table names as **metric tables**.

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

`http://`*`myhost`*`:`*`myport`*`/dmsoc4j/AggreSpy`

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

Figure 2–1 shows a sample AggreSpy response. The AggreSpy response shows two frames, one containing a list of metric tables on the left, and one showing the selected performance metric with current values on the right.

AggreSpy provides navigation and display options, including:

- Access additional metric tables using the links in the left frame.
- Sort rows in the metric tables by clicking on the column headings.
- Format the display in Raw or XML format by clicking on the Raw or XML links.

AggreSpy caches performance data so the AggreSpy metric values reported may not always contain the most current data. AggreSpy refreshes metrics based on how often data values are accessed (values are refreshed at most every 15 seconds).

Note: 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 metric tables may not contain data or may be shown as blanks. If you wait a short time, and then refresh the display, the data is available and data values are shown.

AggreSpy URL and Access Control

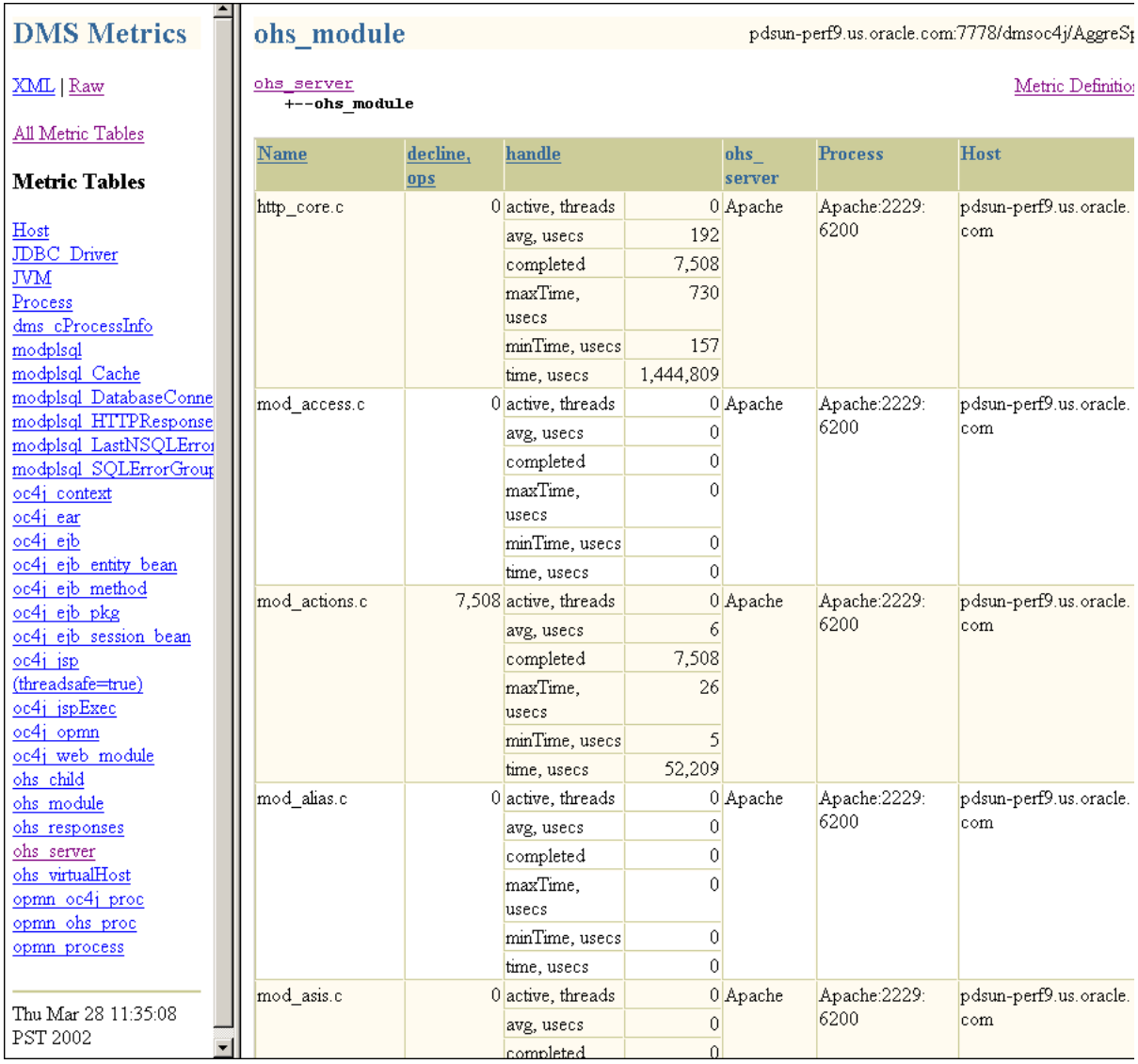
By default, the dmsoc4j/AggreSpy URL is protected, and access to metrics is only allowed from the localhost. If you want to view metrics from a system other than the localhost, then you need to change the access control for dmsoc4j by editing the ORACLE_HOME/Apache/Apache/conf/mod_oc4j.conf file on your Oracle HTTP Server.

The default path for the AggreSpy servlet is dmsoc4j/AggreSpy. If the URL for the dmsoc4j application is changed or the default application is disabled, then you need to update the ORACLE_HOME/Apache/Apache/conf/mod_oc4j.conf file to determine the valid path for accessing the AggreSpy servlet.

See Also:

- *Oracle HTTP Server Administration Guide* for information on configuring mod_oc4j
- *Oracle9i Application Server Security Guide* for information on Oracle HTTP Server access control

Figure 2–1 AggreSpy Performance Metric Display



DMS Metrics

XML | Raw

All Metric Tables

Metric Tables

Host
JDBC_Driver
JVM
Process
dms_cProcessInfo
modplsql
modplsql_Cache
modplsql_DatabaseConne
modplsql_HTTPResponse
modplsql_LastNSQLErro
modplsql_SQLErrorGrou
oc4j_context
oc4j_ear
oc4j_ejb
oc4j_ejb_entity_bean
oc4j_ejb_method
oc4j_ejb_pkg
oc4j_ejb_session_bean
oc4j_jsp
(threadsafe=true)
oc4j_jspExec
oc4j_opmn
oc4j_web_module
ohs_child
ohs_module
ohs_responses
ohs_server
ohs_virtualHost
opmn_oc4j_proc
opmn_ohs_proc
opmn_process

Thu Mar 28 11:35:08 PST 2002

ohs_module pdsun-perf9.us.oracle.com:7778/dmsoc4j/AggreSp

ohs_server
+--ohs_module

Metric Definitio

Name	decline, ops	handle		ohs_ server	Process	Host
http_core.c	0	active, threads	0	Apache	Apache:2229: 6200	pdsun-perf9.us.oracle. com
		avg, usecs	192			
		completed	7,508			
		maxTime, usecs	730			
		minTime, usecs	157			
		time, usecs	1,444,809			
mod_access.c	0	active, threads	0	Apache	Apache:2229: 6200	pdsun-perf9.us.oracle. com
		avg, usecs	0			
		completed	0			
		maxTime, usecs	0			
		minTime, usecs	0			
		time, usecs	0			
mod_actions.c	7,508	active, threads	0	Apache	Apache:2229: 6200	pdsun-perf9.us.oracle. com
		avg, usecs	6			
		completed	7,508			
		maxTime, usecs	26			
		minTime, usecs	5			
		time, usecs	52,209			
mod_alias.c	0	active, threads	0	Apache	Apache:2229: 6200	pdsun-perf9.us.oracle. com
		avg, usecs	0			
		completed	0			
		maxTime, usecs	0			
		minTime, usecs	0			
		time, usecs	0			
mod_asis.c	0	active, threads	0	Apache	Apache:2229: 6200	pdsun-perf9.us.oracle. com
		avg, usecs	0			
		completed	0			

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. 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` is located in the `$ORACLE_HOME/bin` directory.

> **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.

Access Control for dmstool

By default, the `dmstool` only can obtain metrics when it is run from the localhost. If you want to view metrics from an Oracle HTTP Server running on a remote host, then you need to use `dmstool` with the `-a` option and change the access control for `/dms0` by editing the `$ORACLE_HOME/Apache/Apache/conf/httpd.conf` file on your Oracle HTTP Server.

Table 2–2 dmstool Command-line Options

Option	Description

–a[ddress] *host*[:*port*]	By default, without the -a option, dmstool gets metrics from the localhost. When the Oracle HTTP Server is running on the same system as dmstool, the –a option is not required.
	
	You can specify –a multiple times on the command line to monitor a cluster.
	
	host is the domain name or IP address of the host on which the Oracle HTTP Server is running.
	
	port specifies the OPMN request port that supplies the metrics. The request port is specified in $ORACLE_HOME/opmn/conf/opmn.xml, for example:
	
	`<notification-server>`
	`<port local="6100" remote="6200" request="6003"/>`
	`<log-file path="/private/oracle/opmn/logs/ons.log" level="3"/>`
	`</notification-server>`

–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.
–dump	Using dmstool with the -dump option reports all the metrics from an Oracle9*i*AS instance on the standard output. Oracle recommends that you run with the -dump option periodically, such as every 10 to 15 minutes, to capture and save a record of performance data for your Oracle9*i*AS server.
–i[nterval] *secs*	Specifies the number of seconds to wait between metric retrievals. The default is 1 second. 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]	Generates a list of all metrics available. 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, "Oracle9iAS Performance Metrics" or run AggreSpy for a list of metric table names.

Using dmstool to List the Names of All Metrics

Every Oracle9*i*AS 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 Oracle HTTP Server:

```
% dmstool –list
```

This command displays a list of the available metrics.

See Also: Appendix A, "Oracle9iAS Performance Metrics"

Using dmstool to Report 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 full name of the metric that shows the JVM uptime:

   ```
   % dmstool -list | grep JVM/upTime.value
   /myhost/OC4J:3000:6003/JVM/upTime.value
   ```

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

   ```
   % dmstool /myhost/OC4J:3000:6003/JVM/upTime.value
   Tue Apr 02 16:27:32 PST 2002
   /myhost/OC4J:3000:6003/JVM/upTime.value       23991009 msecs
   ```

Suppose you are trying to balance the load on your host across two OC4J processes and you want to monitor the number of requests handled by each OC4J over time. If you generate a list of the available metrics using the `dmstool –list` command, and search the list for information on OC4J, you should find metric names such as:

```
/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/ojsp/WEBs/processRequest.completed
```

This `dmstool –list` output shows that the site contains two OC4J processes. The OC4J listening on AJP port 3000 is running an application called `default`, while the OC4J listening on port AJP 3001 is running an application called `ojsp`. JSPs have been accessed in the `ojsp` application but not in the default application.

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 command, supplying multiple 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, as specified with the –c option, while collecting data at intervals of 60 seconds:

```
                Mon Nov 19 17:13:01 PDT 2001
/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 2001
/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 reports all the metrics from an Oracle9*i*AS instance to the standard output.

The following command displays all available metrics and metric values on the Oracle HTTP Server:

```
% dmstool –dump
```

Oracle recommends that you run `dmstool` with the `-dump` option periodically, such as every 10 to 15 minutes, to capture and save a record of performance data for your Oracle9*i*AS server. 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 View Metrics on a Remote Oracle9*i*AS System

Using `dmstool` with the `-a` option reports all the metrics from a remote Oracle9*i*AS instance.

The following command displays all available metrics and metric values on the Oracle HTTP Server for the given address, as specified with the `–a` option:

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

3

Monitoring Oracle HTTP Server

This chapter discusses how to monitor Oracle HTTP Server performance. Obtaining performance data can assist you in tuning Oracle9*i*AS or in tuning and debugging applications with performance problems.

This chapter contains the following topics:

Monitoring Oracle HTTP Server with Oracle Enterprise Manager

The Oracle HTTP Server is a central and important part of most Oracle9*i*AS 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 Oracle9*i*AS performance issues.

This section covers the following topics:

- Assessing the Oracle HTTP Server Load with Oracle Enterprise Manager
- Investigating Oracle HTTP Server Errors with Oracle Enterprise Manager
- Categorizing Oracle HTTP Server Problems with Oracle Enterprise Manager

Assessing the Oracle HTTP Server Load with Oracle Enterprise Manager

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 Oracle9*i*AS 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 information provides a picture of overall site performance; however if Oracle9*i*AS 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 1-9

Oracle Enterprise Manager provides Oracle HTTP Server performance data in the following categories:

- Status Metrics
- Response and Load Metrics
- Module Metrics

See Also:

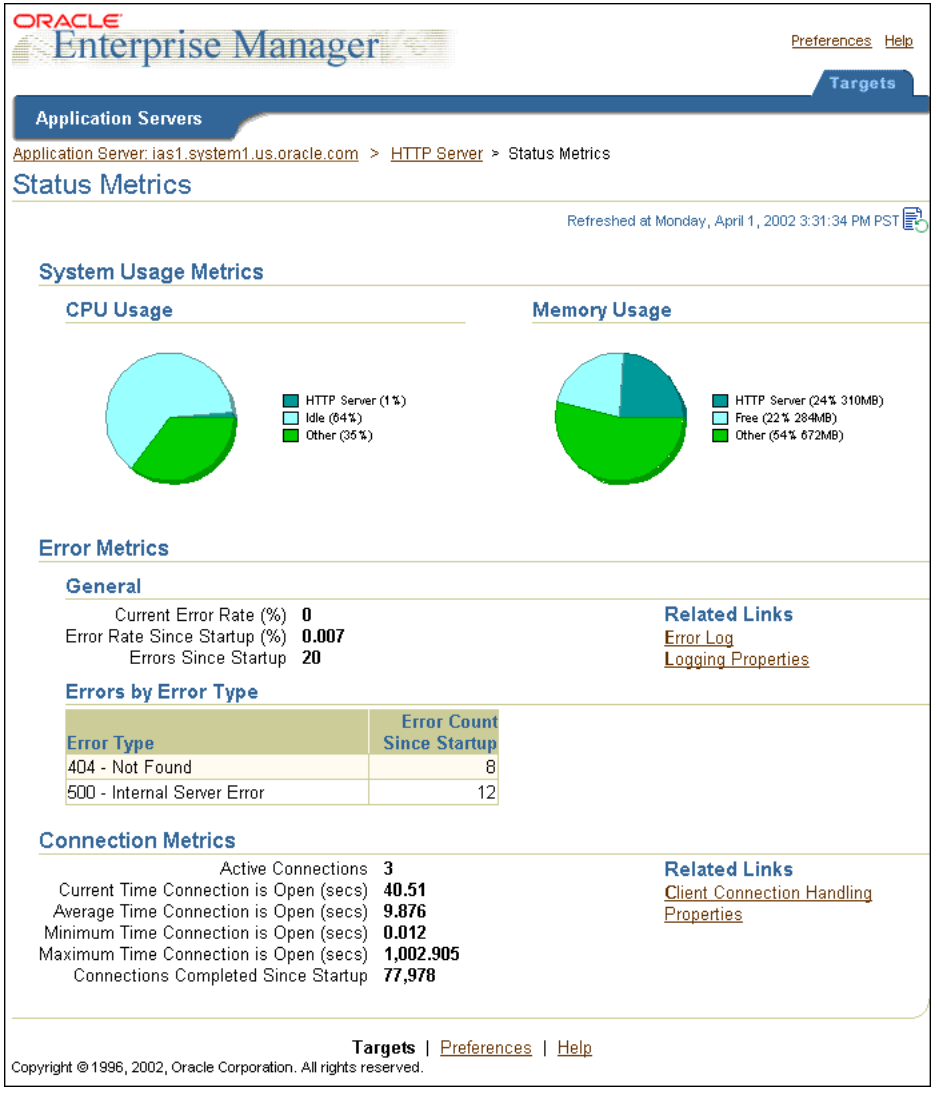
- Chapter 7, "Optimizing Web Cache" for information on optimizing Oracle9*i*AS Web Cache in Oracle9*i*AS
- *Oracle9iAS Web Cache Administration and Deployment Guide* for further details on Oracle9*i*AS Web Cache
- *Oracle9i Application Server Administrator's Guide* for information on using Enterprise Manager with Oracle9*iAS*

Status Metrics

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

Figure 3–1 shows the Enterprise Manager HTTP Server status metrics page.

Figure 3–1 Oracle Enterprise Manager Status Metrics Page



Response and Load Metrics

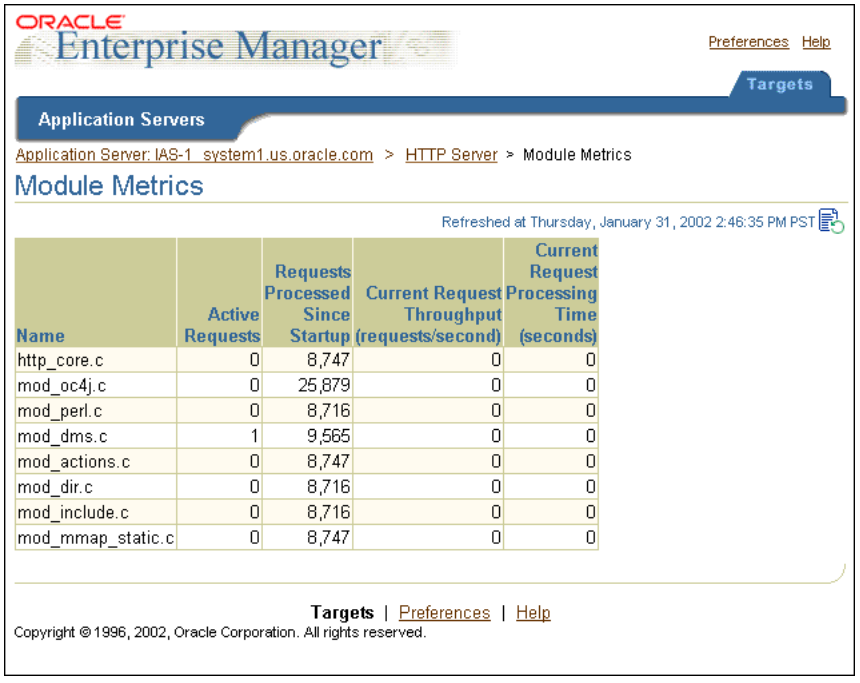
Figure 3–2 shows the Oracle Enterprise Manager Response and Load Metrics page. This page shows values for Oracle HTTP Server Active Requests and Request Throughput, and reports the average, minimum, and maximum processing time for requests. The values on the Response and Load Metrics page can help you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

Figure 3–2 Oracle Enterprise Manager Response and Load Metrics

Module Metrics

Figure 3–3 shows the Oracle Enterprise Manager 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 Oracle Enterprise Manager Module Metrics Page



Investigating Oracle HTTP Server Errors with Oracle Enterprise Manager

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 Oracle9*i*AS 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 Oracle Enterprise Manager you can view the Error Metrics on the HTTP status page, as shown in Figure 3–1. Error Metrics include the current error rate, which is the number of errors occurring in 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 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 a large numbers of Not Found errors, such as a number that is greater than 1% of the total requests.

To investigate errors in more detail, such as any reported internal errors, examine the error log by selecting the Error Log link under the Related Links heading. Using the error log, you should be able to determine more information about the URIs that are causing specific errors.

Categorizing Oracle HTTP Server Problems with Oracle Enterprise Manager

If you notice a performance problem on the Oracle HTTP Server, then where 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.

Oracle Enterprise Manager can help you to categorize performance problems. You can identify where requests are being processed, or where a large percentage of request processing time is concentrated. Using Oracle Enterprise Manager 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 Oracle9*i*AS performance by module.

When viewing the Module Metrics, note the following:

1. The `http_core.c` module handles every request for static pages. If Oracle9*i*AS Web Cache is enabled, then use of `http_core.c` should be reduced. When you are using Oracle9*i*AS Web Cache, you should monitor requests processed by the `http_core.c` module to make sure that Oracle9*i*AS 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 forwarded to OC4J through the `mod_oc4j.c` module. You should then drill down to review the information available for the OC4J of interest. Oracle Enterprise Manager 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 values provide 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 Oracle9*i*AS performance issues by virtual host.

Figure 3–4 Oracle Enterprise Manager 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 Oracle9*i*AS system experiences delays or deadlocks, you may need to analyze the Oracle HTTP Server child server processes. The Process Details page available from Related Links section on the Response and Load Metrics page shows the Process ID for each active Oracle HTTP Server child process. Viewing this information 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.

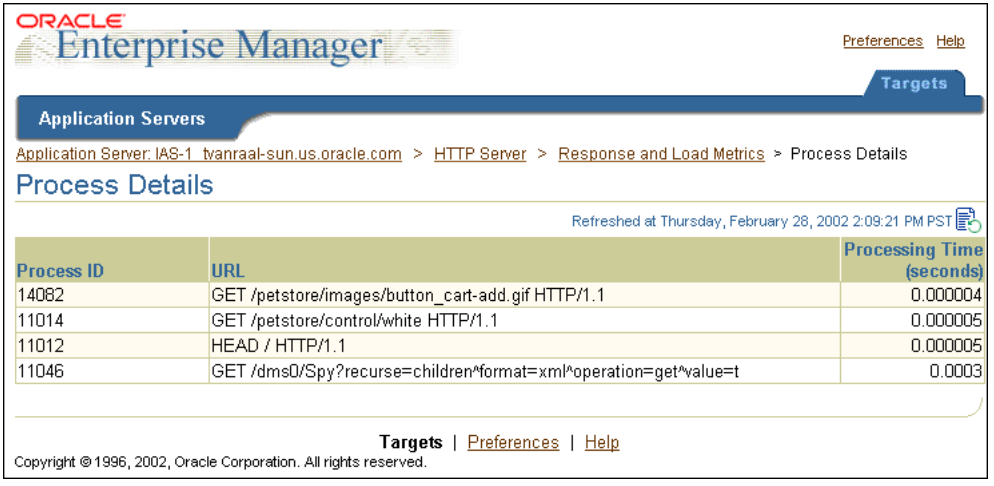
Figure 3–5 shows a Process Details page with Oracle HTTP Server child server information.

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 set with the `TimeOut` directive.

> **See Also:** *Oracle HTTP Server Administration Guide* for information on the `TimeOut` directive in Chapter 4, "Managing the Network Connection"

Figure 3–5 Oracle Enterprise Manager 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 Oracle9*i*AS 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 Oracle9*i*AS 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 Oracle9*i*AS 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 Oracle9*i*AS 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 1-9

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` metrics. Using `AggreSpy`, you can view the `ohs_server` metrics by choosing the `ohs_server` metric table in the left pane of the `AggreSpy` window.

Example 3–1 shows AggreSpy output for the ohs_server metric table using the Raw format.

Example 3–1 Overall HTTP Server Metrics Report

```
 <DMSDUMP version='2.0' timestamp='1017345371143 (Thu Mar 28 11:56:11 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
/pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    internalRedirect.count:7418 ops
    numMods.value:45
    handle.maxTime:22205524 usecs
    handle.minTime:2 usecs
    handle.avg:14274 usecs
    handle.active:2 threads
    handle.time:997159521 usecs
    handle.completed:69858
    request.maxTime:22206941 usecs
    request.minTime:602 usecs
    request.avg:31537 usecs
    request.active:1 threads
    request.time:1033442848 usecs
    request.completed:32769
    connection.maxTime:1002008298 usecs
    connection.minTime:7254 usecs
    connection.avg:258721053 usecs
    connection.active:3 threads
    connection.time:152386700540 usecs
    connection.completed:589
    childFinish.count:0 ops
    childStart.count:11 ops
    lastConfigChange.value:1017260765
    busyChildren.value:1
    readyChildren.value:10
    numChildren.value:11
    responseSize.value:903136783
    error.count:1 ops
    post.count:0 ops
    get.count:32769 ops
</statistics>
</DMSDUMP>
```

The metric table shown in Example 3–1 groups metrics into categories, including `handle`, `request`, and `connection`. The individual metric names in each category have the form *name.metric*, for example, `connection.time`. The metrics in these three categories 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 at the top level, 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.

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

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

These performance metrics indicate how many Oracle HTTP Server child servers are in use, and how many of them are actively processing requests. The data in Example 3–1 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 starts new child servers automatically. 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 the count of 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 3–1, more than 11 child servers have been started and 0 finished.

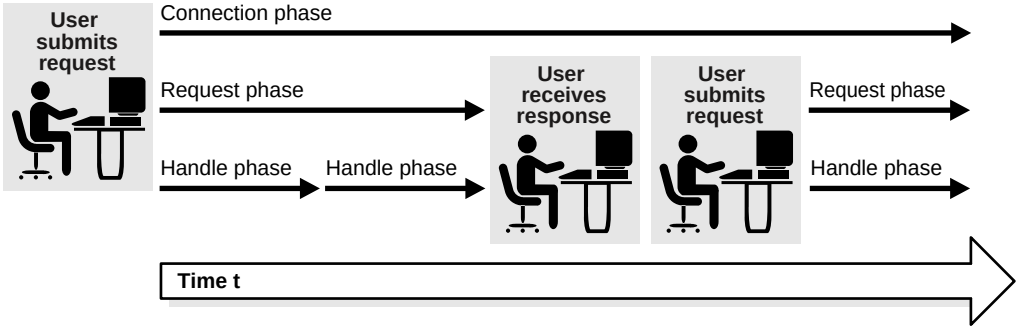
The `childStart.count` and `childFinish.count` metric values could indicate that the instantaneous load for the Oracle HTTP Server exceeded the current load and also exceeded the range assumed by the default Oracle HTTP Server configuration parameters. When the count of child servers started and the count of child servers finished are both large, this could indicate that the Oracle HTTP Server could benefit by tuning the values of 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 Web Cache" for information on optimizing Oracle9*i*AS Web Cache in Oracle9*i*AS
- Appendix A, "Oracle9iAS Performance Metrics"
- *Oracle9iAS Web Cache Administration and Deployment Guide* for further details on Oracle9*i*AS Web Cache
- *Oracle HTTP Server Administration Guide* for information on Oracle HTTP Server configuration parameters related to starting and stopping of 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 Oracle9*i*AS 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 3–1 includes the `ohs_server` metrics that provide an overview of error activity. The `error.count` metric is incremented 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 3–2 shows the AggreSpy report for the ohs_responses metric table in Raw format.

Example 3–2 HTTP Server Responses Metrics (ohs_responses Metric Table)

```
<DMSDUMP version='2.0' timestamp='1017345294216 (Thu Mar 28 11:54:54 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
 /pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Responses [type=ohs_
responses]
  SvrErr_Not_Extended_510.count:        0 ops
  .
  .
  .
  CltErr_Method_Not_Allowed_405.count:  0 ops
  CltErr_Not_Found_404.count:   29 ops
  Redirect_NotModified_304.count:       23 ops
  Success_Created_201.count:    0 ops
  Success_OK_200.count: 10103 ops
  Info_Processing_102.count:    0 ops
 </statistics>
</DMSDUMP>
```

Example 3–2 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:** *Oracle HTTP Server Administration Guide* for information on the Oracle HTTP Server access_log and error_log files

Categorizing Oracle HTTP Server Performance Problems with Built-in Metrics

If you notice a performance problem on the Oracle HTTP Server, then where 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 3–3 shows `AggreSpy` raw format output for the `ohs_module` metric table.

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

```
<DMSDUMP version='2.0' timestamp='1017345223482 (Thu Mar 28 11:53:43 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
/pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Modules [type=n/a]
     /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Modules/mod_mmap_
static.c [type=ohs_module]
      handle.maxTime:182 usecs
      handle.minTime:3 usecs
      handle.avg:5 usecs
      handle.active:0 threads
      handle.time:38942 usecs
      handle.completed:7562
      decline.count:7562 ops
```

```
 ...
/Apache/Modules/mod_cgi.c [type=ohs_module]
 ...
 handle.avg:  199730 usecs
 ...
/Apache/Modules/mod_perl.c [type=ohs_module]
 handle.maxTime:      768041 usecs
 ...
/Apache/Modules/mod_fastcgi.c [type=ohs_module]
 ...
 handle.avg:  5866 usecs
 ...
/Apache/Modules/mod_oc4j.c [type=ohs_module]
 handle.maxTime:      33676386 usecs
 handle.minTime:      165 usecs
 handle.avg:  5488 usecs
 handle.active:       0 threads
 handle.time: 317776833 usecs
 handle.completed:    57902
 decline.count:       0 ops
 ...
/Apache/Modules/http_core.c [type=ohs_module]
 ...
 handle.completed:    93535
 ...
```

When viewing the Module Metrics, note the following:

1. The `http_core.c` module handles every request for static pages. If Oracle9*i*AS Web Cache is enabled, then use of `http_core.c` should be reduced. If Oracle9*i*AS Web Cache is enabled the you should monitor the `http_core.c` metrics to make sure that Oracle9*i*AS 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 3–3, 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)
   ```

```
= (317776833 - 165 - 33676386)/ (57902 - 2)
= 4907 milliseconds
```

3. Viewing the `ohs_module` metric table may show you that many requests were forwarded to OC4J through the `mod_oc4j.c` module. Oracle9*i*AS 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 3–4 shows the `AggreSpy` raw format output for the `ohs_virtualHost` metric table.

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

```
<DMSDUMP version='2.0' timestamp='1017345119223 (Thu Mar 28 11:51:59 PST 2002)'
id='3000' name='pdsun-perf9.us.oracle.com:7778'>
<statistics>
 /pdsun-perf9.us.oracle.com [type=Host]
  /pdsun-perf9.us.oracle.com/Apache:2229:6200 [type=Process]
   /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache [type=ohs_server]
    /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Virtual_Hosts [type=n/a]
     /pdsun-perf9.us.oracle.com/Apache:2229:6200/Apache/Virtual_
Hosts/pdsun-perf9.us.oracle.com [type=ohs_virtualHost]
      responseSize.value:       0
      request.maxTime:  0 usecs
      request.minTime:  0 usecs
      request.avg:      0 usecs
      request.active:   0 threads
      request.time:     0 usecs
      request.completed:        0
</statistics>
</DMSDUMP>
```

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 Oracle9*i*AS 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 3–5 shows the `AggreSpy` raw format output for the `ohs_child` metric table.

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 3–5, the top entry has been executing for 117 million microseconds, which is nearly two minutes. The `ohs_child` metrics include the URL associated with the request and the process identifier for each Oracle HTTP Server child server listed.

Example 3–5 Drill Down to Investigate Activity per Child Server

```
/Apache [type=ohs_server]
      /Apache/Children [type=n/a]
       /Apache/Children/Child00 [type=ohs_child]
        time.value:  117045690 usecs
        pid.value:   2466
        status.value:        writing
        url.value:   GET /cgi-bin/deadlock HTTP/1.1
        slot.value:  2
       /Apache/Children/Child01 [type=ohs_child]
        time.value:  5 usecs
        pid.value:   2469
        status.value:        writing
        url.value:   GET /dms0/Spy?name=/Apache/Children HTTP/1.1
        slot.value:  5
       /Apache/Children/Child02 [type=ohs_child]
        time.value:  4 usecs
        pid.value:   2465
        status.value:        keepalive
        url.value:   HEAD / HTTP/1.1
        slot.value:  1
       /Apache/Children/Child03 [type=ohs_child]
        time.value:  2 usecs
        pid.value:   7591
```

```
status.value:        writing
url.value:   GET /fcgi-bin/echo HTTP/1.0
slot.value:  8
```

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 Administration Guide* for information on the `TimeOut` directive in Chapter 4, "Managing the Network Connection"

4

Monitoring OC4J

This chapter discusses how to monitor the performance of Oracle9*i*AS Containers for J2EE (OC4J). Obtaining performance data can assist you in tuning Oracle9*i*AS or in tuning and debugging applications with performance problems.

This chapter contains the following topics:

Monitoring OC4J With Oracle Enterprise Manager

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

- Monitoring OC4J Instances With Oracle Enterprise Manager
- Monitoring J2EE Applications with Oracle Enterprise Manager

Monitoring OC4J Instances With Oracle Enterprise Manager

Before analyzing OC4J performance, make sure that your OC4J instance is running. Figure 4–1 shows an Oracle Enterprise Manager display that shows the status for a selected OC4J instance (the Up under the heading General indicates the OC4J instance is running).

Figure 4–1 shows an Oracle Enterprise Manager OC4J instance page. Oracle Enterprise Manager provides overall OC4J performance data for active OC4J instances. The overall performance data includes OC4J performance data collected in the following categories:

- General
- Status
- Monitoring OC4J
- Response for EJBs
- JDBC Usage

Figure 4–1 Oracle Enterprise Manager OC4J Instance Display

General

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

Status

The Oracle Enterprise Manager OC4J Status information shows the CPU usage, memory usage, and heap usage for the OC4J instance.

Response for Servlets and JSPs

The Oracle Enterprise Manager 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.

Response for EJBs

The Oracle Enterprise Manager OC4J Response information for EJBs shows the number of active EJB methods and the EJB method execution rate.

JDBC Usage

The Oracle Enterprise Manager 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.

Monitoring J2EE Applications with Oracle Enterprise Manager

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.

For each J2EE application deployed on Oracle9*i*AS, you can view performance information in several categories.

Figure 4–2 shows the Oracle Enterprise Manager display for the sample petstore application.

Figure 4–2 Oracle Enterprise Manager J2EE Application Metrics

Figure 4–2 shows the available Oracle Enterprise Manager 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 Oracle Enterprise Manager J2EE application General information provides an indication of whether the application is loaded or not in the status field, an 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 Oracle Enterprise Manager 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 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 Oracle Enterprise Manager J2EE application Response information for EJBs shows the number of active EJB methods and the EJB method execution rate.

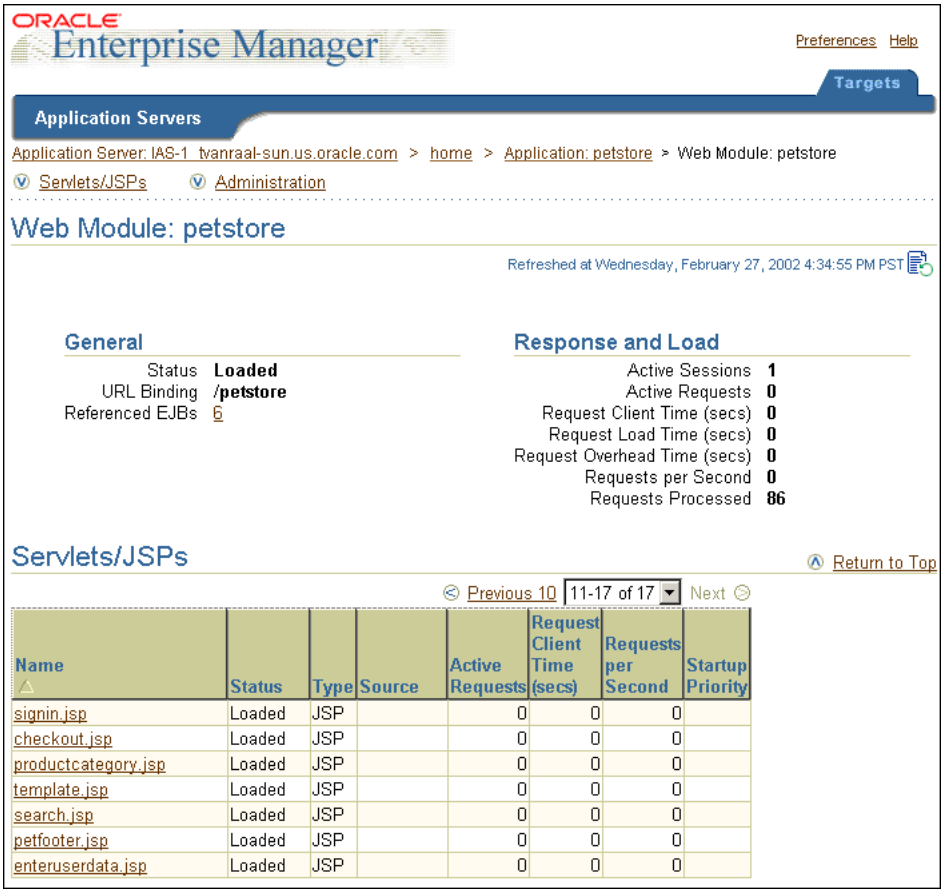
For more detail on this information or to drill down to specific Servlets and JSPs, use the links in the EJB Modules table.

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 petstore 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 Oracle Enterprise Manager J2EE Application Web Module Metrics

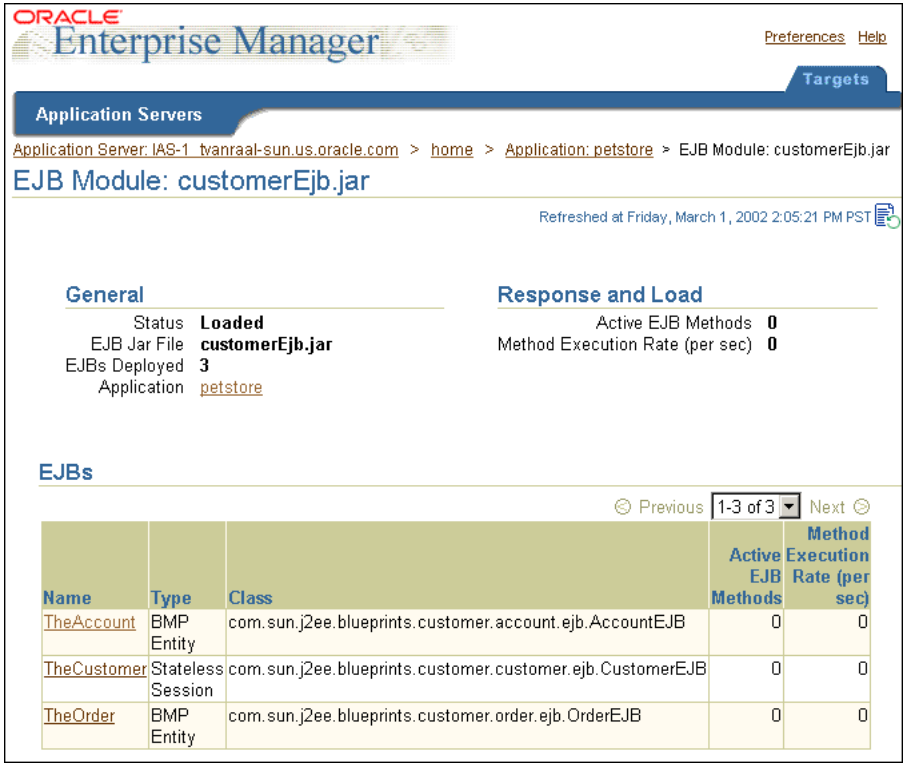


EJB Modules Table

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 EJB Module page.

Figure 4–4 Oracle Enterprise Manager EJB Module Page



Monitoring OC4J With Built-in Performance Metrics

You can use the Oracle9*i*AS built-in performance metrics to analyze OC4J and J2EE application performance. Before you attempt to monitor OC4J performance, verify that the OC4J home instance, that is installed by default with Oracle9*i*AS, 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 Oracle9*i*AS Containers for J2EE.

From the OC4J Welcome page you can access the samples for JSPs and servlets. If you do not have active J2EE applications that generate requests on your OC4J instance, then you can use your browser to request the 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 built-in performance metrics.

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

```
http://myhost:myport/dmsoc4j/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 Oracle9*i*AS 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 and other components, checking collected metrics against your design and deployment assumptions. You should check these assumptions with single browser client scenarios, with simulated multi-user 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, and EJB methods 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 processes in the 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 2-4
- "Viewing Performance Metrics Using dmstool" on page 2-7
- Chapter 6, "Optimizing J2EE Applications In OC4J"
- Appendix A, "Oracle9iAS 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 Oracle9*i* Application Server.

This chapter contains:

TCP Tuning Parameters

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 Recommended TCP Parameter Settings for Solaris

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_close_wait_interval`	60000	Parameter name in Solaris release 2.6.
`tcp_time_wait_interval`	60000	Parameter name in Solaris release 2.7 or later. 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	

Table 5–2 TCP Parameter Settings for HP-UX

Parameter	Scope	Default Value	Tuned Value	Comments
tcp_rexmit_interval_max	ndd/dev/tcp	60,000	60,000	
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	
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

Tuning a Running System

There is no `sysctl` application for changing kernel values. You can change the kernel values with an editor like VI.

Tuning the Default and Maximum Size

Edit the files listed below 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 above TCP parameter values are set default by a header file in the Linux kernel source directory /LINUX-SOURCE-DIR/include/linux/skbuff.h

These values are default. 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 /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 via a 2Mbit satellite link, measured with netperf.

Setting TCP Parameters

To set the connection table **hash** parameter on Solaris, you must add the following line to your `/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_close_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` on Solaris, add the line shown below to your `/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_close_wait_interval` parameter (renamed `tcp_time_wait_interval` on Solaris 2.7). The default in Solaris 2.x 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 Solaris 2.6:

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

On HP-UX and Solaris 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 Solaris:

```
% /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, Solaris sends only one packet during connection initiation, per the TCP standard. Thus 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:

```
% /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.

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
`MaxClients`	Limit on total number of servers running, that is, limit on the number of clients who can simultaneously connect. If this limit is ever reached, clients are locked out, so it should not be set too low. It is intended mainly as a brake to keep a runaway server from taking the system with it as it spirals down.
`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 a few (such as Solaris) do have notable leaks in the libraries. For these platforms, set 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 towards this limit.
`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
`StartServers`	Number of servers to start initially should be a reasonable ballpark figure. 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

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

Directive	Description
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. If the MaxClients setting is too low, and the limit is reached, clients will be unable to connect.

Tests on a previous release, with static page requests (average size 20K) on a 2 processor, 168 MHz Sun UltraSPARC on a 100 Mbps network 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 above, and on 4 and 6-processor, 336 MHz 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, to act as a throttle on the number of concurrent requests on the server.

How Persistent Connections Can Reduce httpd Process Availability

There are some serious drawbacks to using persistent connections with Oracle HTTP Server. In particular, because httpd processes are single threaded, one client can keep a process tied up for a significant period of time (the amount of time depends on your `KeepAlive` settings). If you have a large user population, and you set your `KeepAlive` limits too high, clients could be turned away because of insufficient httpd deamons.

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.

Logging

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.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.

Secure Sockets Layer

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**. 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 `SSLSessionCacheTimeout` directive in `httpd.conf` determines how long the server keeps a session alive (the default is `300` seconds). The session information is kept in a file. You can specify where to keep the session information using the `SSLSessionCache` directive; the default location is the `$ORACLE_HOME/Apache/Apache/logs/` directory. The file can be used by multiple Oracle HTTP Server processes.

The duration of an SSL session is unrelated to the use of HTTP persistent connections.

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 your requests are static and what percentage are dynamic. This is because static pages can be cached by Web Cache. Generally speaking, you want to concentrate your tuning effort on dynamic pages because they are normally more 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 7, "Optimizing Web Cache" for further details regarding Web Cache
- Chapter 3, "Monitoring Oracle HTTP Server"

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 Oracle9*i*AS Containers for J2EE (OC4J) and the average response time experienced by the user. If the time is not being spent actually doing the work, then it is probably being spent in transport. If you notice a large discrepancy, please consider the performance guidelines specified in the section, "Configuring Oracle HTTP Server Directives" on page 5-9.

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 it is upon the first request that a database connection must be opened. Later requests can make use of the established connection. To get a better estimate of the average service time for a PL/SQL module, recalculate the average as in the following:

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

The values would be:

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

6

Optimizing J2EE Applications In OC4J

This chapter provides guidelines for improving the performance of Oracle9*i*AS Containers for J2EE (OC4J) applications in Oracle9*i* Application Server.

This chapter contains:

Note: This chapter describes using Oracle Enterprise Manager 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 for some Oracle9*i*AS 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 6-3
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 6-8
Managing Concurrency and Limiting Connections	See "Limiting HTTP Connections" on page 6-34
Load Balancing	See "Configuring Multiple OC4J Processes" on page 6-36
Balancing Applications	See "Balancing Applications Across OC4J Instances" on page 6-37
Database Monitoring and Tuning	See "Database Monitoring and Tuning" on page 6-37

Improving J2EE Application Performance by Configuring OC4J Instance

Tuning OC4J configuration options allows you to 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 Options for OC4J Processes
- Setting Up Data Sources – Performance Issues

See Also:

Chapter 3, "Monitoring Oracle HTTP Server"

Chapter 4, "Monitoring OC4J"

Setting Java Options for OC4J Processes

When running Oracle9*i*AS, 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.

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

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 Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-6, 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 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 Oracle Enterprise Manager 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, include calls in your program to the `Runtime.getRuntime().totalMemory()` and `Runtime.getRuntime().freeMemory` methods in the `java.lang` package. Subtract free memory from total memory; the difference is the amount of heap that the application consumed.

> **See Also:** You can find detailed information about JVM options and their impact on performance on the JVM vendor's web sites.

Setting the Server Option for OC4J Processes

Depending on the particular J2EE application, setting the command line option `-server` for the JVM running OC4J may improve performance (the JVM runs in one of two modes set with the two related options, `-client` and `-server`, the default value is `-client`). To set this option, use the procedures outlined in "Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-6, and specify the `-server` Java option.

The `-server` option selects the server VM instead of using the default client VM. The client and the server VMs are similar, except that the server VM is specially tuned to maximize peak operating speed. It is intended for executing long-running server applications, for which having the fastest possible operating speed is generally more important than having a fast startup time or a smaller runtime memory footprint. The client VM, the default without using the `-server` option starts up faster and requires a smaller memory footprint than the server VM.

Note: The `-server` option must be specified first, before all other Java options.

Setting the 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 Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-6, 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 Concurrentio Option for OC4J Processes

Depending on the particular J2EE application and JDK version, changing the setting of the command line option `-Xconcurrentio` for the JVM running OC4J may improve performance. To set this option, use the procedures outlined in "Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options" on page 6-6, and specify the `-Xconcurrentio` Java option.

This option generally helps programs with many threads. The main feature turned on with `-Xconcurrentio` is to use LWP based synchronization instead of thread based synchronization. For certain applications, with JDK 1.3.1, this option increases speed up by over 40%. See the following site for more information,

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

Using the `-Xconcurrentio` option, it is important to compare results for your application without the option. In some tests, results have been mixed, using the option results in a speedup for some applications, but for other applications or JDK versions, the performance degraded with this option. See the following site,

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

Note: In JDK 1.4, LWP based synchronization is the default, but setting `-Xconcurrentio` can still improve performance in JDK 1.4 since it turns on additional internal options.

Using Oracle Enterprise Manager to Change OC4J JVM Command Line Options

To change the Java command line options for an OC4J Instance, go to the home page for the OC4J Instance 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.

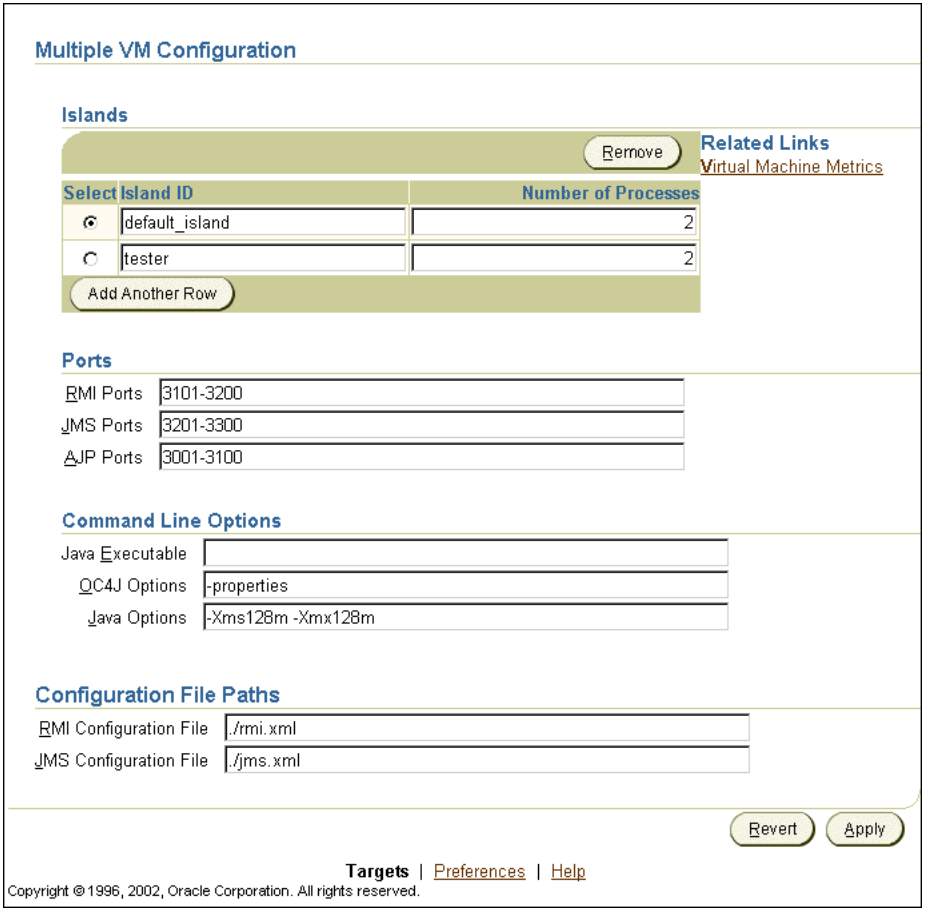
For example, enter the following to set the JVM heap sizes to 128 Megabytes:

```
-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 Oracle Enterprise Manager



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
- Using Oracle Enterprise Manager 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 Oracle9iAS" on page 6-27
- Chapter 4, "Data Sources Primer" in *Oracle9iAS Containers for J2EE User's Guide*
- Chapter 15, "Data Sources" in *Oracle9iAS Containers for J2EE Services Guide*
- *Oracle9iAS Containers for J2EE Enterprise JavaBeans Developer's Guide and Reference*

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 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 6–2 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 6-13
- Chapter 15, "Data Sources" in *Oracle9iAS Containers for J2EE Services Guide*

Table 6–2 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 6-11

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.

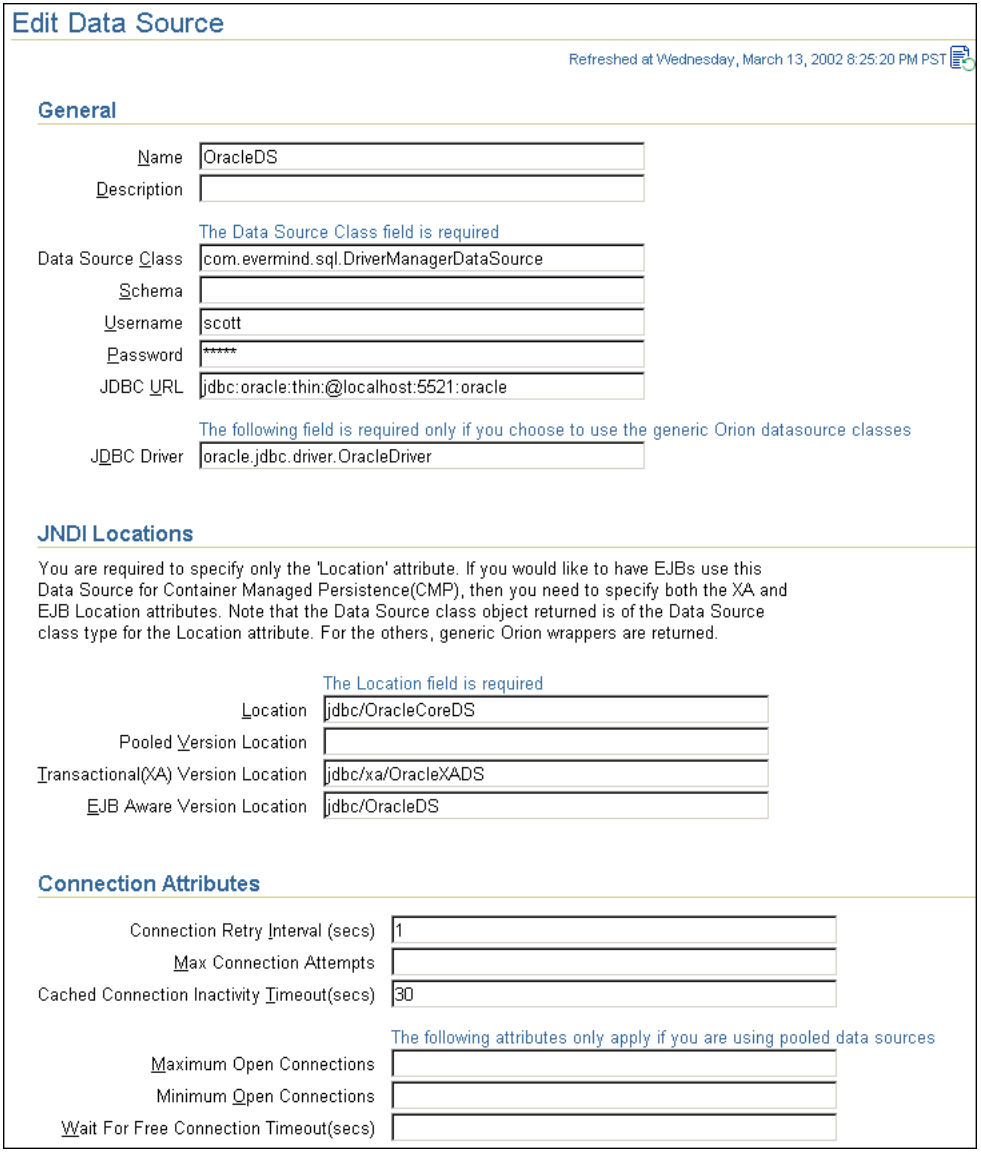
Using Oracle Enterprise Manager to Change Data Source Configuration Options

Figure 6–2 shows the Oracle Enterprise Manager configuration page that lets you view or modify a data source. This page is available in Oracle Enterprise Manager 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).

Oracle Enterprise Manager 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 Oracle Enterprise Manager 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 Oracle Enterprise Manager Data Sources Configuration Page



Improving Servlet Performance in Oracle9*i*AS

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

This section cover the following:

Loading Servlet Classes at Startup

By default, OC4J loads a servlet when the first request is made. OC4J also allows you to 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.

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

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

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

Using Oracle Enterprise Manager you can also specify that OC4J load for an entire Web Module on startup. To specify that a web module is to be loaded on startup, select the Website Properties page for an OC4J Instance and then use the Load on Startup checkbox.

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
- Watch for Abnormal Session Usage
- 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–11 in Appendix A, "Oracle9iAS Performance 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 Oracle Enterprise Manager, or by viewing the application's web module metrics.

> **See Also:** Table A–11 in Appendix A, "Oracle9iAS Performance 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 6-16

Watch for Unused Sessions

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.time` and `sessionActivation.completed` and `sessionActivation.active`.

> **See Also:** Table A–12 in Appendix A, "Oracle9iAS Performance Metrics"

Watch for Abnormal Session Usage

This example shows an application that creates sessions, but never uses them:

To provide an 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 it causes memory or CPU consumption problems.

A well-tuned application shows `sessionActivation.active` with a value that is less than `sessionActivation.completed` before the session time out. 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 when more than 500 have been created and completed indicates that sessions are being invalidated after use.

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 6-16

Improving JSP Performance in Oracle9*i*AS

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.

Oracle9*i*AS 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 Oracle9*i*AS Web Cache. This supports the use of the HTTP-level cache, maintained outside the application, that provides very fast cache operations. Oracle9*i*AS 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 allows you to capture intermediate results of JSP and servlet execution, and subsequently reuse these cached results in other parts of the Java application logic.

See Also:

- "Improving JSP Performance in Oracle9iAS" on page 6-20
- Chapter 5, "Key Considerations" in *Oracle9iAS Containers for J2EE Servlet Developer's Guide*
- *Oracle9iAS 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`.

See Also:

- *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* in the Oracle9*i* Application Server documentation library for information on JSP configuration parameters.
- *Oracle9iAS Containers for J2EE Servlet Developer's Guide* in the Oracle9*i* Application Server documentation library 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 6–3 shows the supported settings for `main_mode`.

Table 6–3 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.

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 session objects created. 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 30 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:

```
HtttpSession 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:

- *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* for information on sessions
- *Oracle9iAS 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: *Oracle9iAS Containers for J2EE Support for JavaServer Pages Reference* in the Oracle9*i* Application Server documentation library

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 `jsp:include` workaround mentioned above, because the page implementation class must load the resource file whenever the class is loaded.

Improving EJB Performance in Oracle9iAS

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:

Setting server.xml Configuration Parameters for EJBs

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

Setting the Transaction Configuration Timeout

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. The `transaction-config` timeout is not an EJB specific timeout, but affects all transactions which use EJBs. This configuration parameter specifies the maximum time taken for a transaction to finish before it can get rolled back due to a timeout. This parameter applies to all transactions on the OC4J Instance. The default value for this parameter is 60000 milliseconds (60 seconds).

For example, the following `server.xml` sets the `transaction-config` timeout parameter to 30 seconds:

```
<transaction-config timeout="30000"/>
```

Increase this value if your applications that use transactions are getting transaction timeout errors, or if your transactions may be longer than 60 seconds (including waiting for connections set by `wait-timeout` in `datasources.xml`).

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 6-13
- "Configuring Parameters that Apply for All EJBs" on page 6-28

Setting OC4J Specific Configuration Parameters for EJBs

This section covers parameters that you can tune for EJB performance that are specific to OC4J. These parameters are set in the `orion-ejb-jar.xml` file.

This section covers the following topics:

- Configuring Parameters that Apply for All EJBs
- Configuring Parameters for CMP Entity Beans
- Configuring Parameters for BMP Entity Beans
- Configuring Parameters for Session Beans

Configuring Parameters that Apply for All EJBs

Table 6–4 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. The parameters in Table 6–4 are specified in `orion-ejb-jar.xml`.

Table 6–4 EJB Parameters That Apply for All EJB Types

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. Default Values: 90000 milliseconds for entity beans and 0 (forever) for session beans. See Also: "Setting the Transaction Configuration Timeout" on page 6-27

Table 6–4 (Cont.) EJB Parameters That Apply for All EJB Types

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: 3 (for session beans and entity beans) See Also: "Setting the Transaction Configuration Timeout" on page 6-27 See Also: "Setting the Connection Retry Interval in Data Sources" on page 6-13

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 that affect performance.

Table 6–5 lists the entity bean CMP specific parameters.

Table 6–6 describes the supported `locking-mode` parameter values.

Table 6–5 CMP Entity Bean Performance Parameters and Descriptions

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`do-select-before-insert`	Recommend setting to `false` to avoid the extra select before insert which checks if the entity already exists before doing the insert. This will then detect a duplicate, if there is one, during the insert. Default Value: true
`exclusive-write-access`	Default is false for beans with `locking-mode=optimistic` or `pessimistic` and `true` for `locking-mode=read-only`. This parameter corresponds to which commit option is used (A, B or C, as defined in the EJB specification). When `exclusive-write-access = true`, this is commit option A. 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.

Table 6–5 (Cont.) CMP Entity Bean Performance Parameters and Descriptions

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 6–7 for a description of `isolation` options and how they relate to locking modes. Default value:
`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 6–6 for a description of `locking-mode`. See Table 6–7 for a description of `isolation` options and how they relate to locking modes.
`max-tx-retries`	For a description, see Table 6–4
`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. 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.

Note: The following CMP entity bean parameters are not supported in this release:

`max-instances`, `min-instances`, `pool-cache-timeout`

Table 6–6 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 6–7 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 6–7 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 6–7 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 (i.e. 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 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 6–8 lists the entity bean BMP specific parameters.

Table 6–8 BMP Entity Bean Performance Parameters and Descriptions

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`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-tx-retries`	For a description, see Table 6–4

Note: The following BMP entity bean parameters are not supported in this release.

`max-instances`, `min-instances`, `pool-cache-timeout`

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 6–9 lists the stateless session bean specific parameters.

Table 6–10 lists the stateful session bean specific parameters.

Table 6–9 Stateless Session Bean Parameters

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`cache-timeout`	The `cache-timeout` applies for Stateless Session EJBs. This parameter specifies how long to keep stateless sessions cached in the pool. For stateless session beans, if you specify a `cache-timeout`, then at every `cache-timeout` interval, all beans in the pool, of the corresponding bean type, are removed. If the value specified is zero or negative, then the `cache-timeout` is disabled and beans are not removed from the pool. Default Value: 60 (seconds)
`max-tx-retries`	For a description, see Table 6–4

Table 6–10 Stateful Session Bean Parameters

Parameter	Description
`call-timeout`	For a description, see Table 6–4
`timeout`	The `timeout` applies for Stateful Session EJBs. If the value is zero or negative, then all timeouts are disabled. The timeout parameter is an inactivity timeout for stateful session beans. Every 30 seconds the pool clean up logic is invoked. Within the pool clean up logic, only the sessions that timed out, by passing the `timeout` value, are deleted. Adjust the `timeout` based on your applications use of stateful session beans. For example, if stateful session beans are not removed explicitly by your application, and the application creates many stateful session beans, then you may want to lower the `timeout` value. If your application requires that a stateful session bean be available for longer than 30 minutes, then adjust the timeout value accordingly. Default Value: 30 (minutes)
`max-tx-retries`	For a description, see Table 6–4

Using Multiple OC4Js and Limiting Connections

This section describes techniques that allow you to improve performance by setting the number of active processes within an OC4J Instance, sending applications to different OC4J Instances, and limiting the number of requests sent to an OC4J Instance.

This section covers the following topics:

- Limiting HTTP Connections
- Configuring Multiple OC4J Processes
- Balancing Applications Across OC4J Instances

Limiting HTTP 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 5-9

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` attributes.

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"/>
```

Table 6–11 describes the `max-http-connections` attributes.

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://optional.redirect.url/page.jsp`, add the following line to `server.xml`:

```
<max-http-connections max-connections-queue-timeout="120" socket-backlog="50"
value="100">http://optional.redirect.url/page.jsp</max-http-connections>
```

Table 6–11 Setting max-http-connections Attributes

Attribute	Description
`max-connections-queue-timeout`	Specifies the number of seconds to wait for an available connection if the maximum connections have been reached before returning either server busy or redirect message. Default Value: 10 (seconds)
`socket-backlog`	Specifies the number of connections that can be queued up before denying connections at the socket level. Default Value: 30
`value`	Specifies the maximum number of connections. Default Value: 100000

See Also: Appendix B, "Additional Information" in the *Oracle9iAS Containers for J2EE User's Guide* for information on the elements in OC4J `server.xml` file.

Configuring Multiple OC4J Processes

Oracle9*i*AS lets you configure multiple OC4J processes in an OC4J Instance. If there are sufficient resources on your system, you may be able to improve performance by configuring multiple OC4J processes for the OC4J Instance that services your J2EE application. Using multiple OC4J processes is the simplest form of load balancing without using Oracle9*i*AS clustering. Configuring an OC4J Instance to allow requests to be served by multiple OC4J processes can reduce or avoid contention and improve scalability. Oracle9*i*AS also supports multi node clustering for load balancing.

The optimal ratio of OC4J processes to CPUs is dependent on characteristics of the application, hardware configuration and JDK being used. But 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 70% of the resources on a four processor system. That statistic indicated that either adding a second OC4J process would be advantageous or that the incoming load was insufficient to utilize more of the machine resources. By monitoring the incoming load, we were able to determine that it was the former, and adding a second process improved 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 four more 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 2-3 CPUs and measure the improvement derived from adding additional processes.

Configuring Multiple OC4J Processes Using Oracle Enterprise Manager

Using Oracle Enterprise Manager you can specify the number of processes in an OC4J Instance using the OC4J Instance level configuration available on the OC4J Instance's home page.

If you configure your OC4J to use load-balancing with multiple OC4J processes running within the same island, and you do not want sessions to be replicated, make sure that the `<distributable/>` property is not set in the application's `web.xml` file.

Note: Setting the `<distributable/>` property in `web.xml` can have significant performance overhead for applications that use sessions, since this configures the application to use session replication with failover when multiple OC4J processes are running within the same island.

See Also: Chapter 3, "Advanced Configuration, Development, and Deployment", in *Oracle9iAS Containers for J2EE User's Guide*

Balancing Applications Across OC4J Instances

It is often beneficial to spread J2EE application load among multiple OC4J Instances, especially when applications are run on a multiprocessor system. Running multiple OC4J Instances generally results in higher **throughput** and shorter **response time**, even on a single-processor host.

If your web site has many different applications deployed which have different requirements and needs, you may want to configure different OC4J Instances, each with the appropriate number of OC4J processes to service the different applications.

To deploy applications to different OC4J Instances, perform the following steps:

1. Create the multiple OC4J Instances.
2. Using the Application Deployment Wizard, for each Instance, deploy the appropriate application to each instance, 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 throughput increases, or the response time decreases.

Database Monitoring and Tuning

To achieve optimal performance in Oracle9*i*AS OC4J J2EE 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.

See Also: *Oracle9i Database Performance Tuning Guide and Reference*

Improving BC4J Performance in Oracle9*i*AS

This section contains tips for improving the maintainability, scalability, and performance of your Oracle Business Components for Java (BC4J) applications.

> **See Also:** *Developing Business Components* under the heading Oracle9*i*AS Business Components for Java in the Oracle9*i*AS documentation library

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 crash, 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.

7

Optimizing Web Cache

This chapter provides guidelines for improving the performance of Oracle9*i*AS Web Cache. It focuses on suggestions about how to size the cache, including the number of CPUs, the amount of memory, the network bandwidth, and the number of network connections.

This chapter contains the following topics:

- Use Two CPUs for Oracle9iAS Web Cache
- Configure Enough Memory for Oracle9iAS Web Cache
- Make Sure You Have Sufficient Network Bandwidth
- Set a Reasonable Number of Network Connections

Note: See the *Oracle9iAS Web Cache Administration and Deployment Guide* for further details.

Use Two CPUs for Oracle9*i*AS Web Cache

Oracle9*i*AS Web Cache is designed to use one or two CPUs. Because Oracle9*i*AS 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 Oracle9*i*AS Web Cache is limited by the available addressable memory. Additional memory can increase performance and scalability. See "Configure Enough Memory for Oracle9iAS Web Cache" on page 7-3 for information about the amount of memory you need.

Oracle9*i*AS Web Cache has three processes: one for the admin server, one for the auto-restart monitor, and one for the cache server.

- The admin server process is used for configuring and monitoring Oracle9*i*AS Web Cache. When the process is idle, it consumes a few CPU cycles, depending on the settings you chose for the Health Monitor. For example, if you choose Every 15 Seconds for a refresh rate on the Health Monitor page, the admin server process uses more CPUs than if you choose Every Minute.
- The auto-restart monitor process checks whether the cache is running. If not, it restarts the cache. The auto-restart monitor uses a minimal amount of CPU.
- The cache server uses two threads: one to manage incoming connections and one to process requests. Because of this, two CPUs dedicated to Oracle9*i*AS Web Cache are optimal.

For a cost-effective way to run Oracle9*i*AS Web Cache, run it on a fast two-CPU computer with lots of memory. See Oracle9*i*AS Web Cache Deployment and Administration Guide for information about various deployment scenarios.

For a Web site with more than one Oracle9*i*AS Web Cache instance, consider installing each instance on a separate two-CPU node, either as part of a cache cluster or as standalone instances. When Oracle9*i*AS 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 limitation than two caches on one four-CPU node.

Of course, if other resources are competing with Oracle9*i*AS 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 Web Cache is optimal, you can also derive a significant performance benefit from Web Cache running on the same node as the rest of the application server.

Configure Enough Memory for Oracle9*i*AS 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 Oracle9*i*AS Web Cache should be set to at least 256 MB. By default, the maximum cache size is set to 50 MB, which is sufficient only for initial post-installation testing.

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 4 KB and how many are larger than 4 KB. Determine the average size of the documents that are larger than 4 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 Oracle9*i*AS Web Cache.

   The amount of memory that Oracle9*i*AS Web Cache uses to store a document depends on the document size:

   - If a document is smaller than 4 kilobytes (KB), Oracle9*i*AS Web Cache uses a buffer of 4 KB to store the HTTP body.
   - If a document is 4 KB or larger, Oracle9*i*AS Web Cache uses buffers of 32 KB to store the HTTP body. For example, if a document is 40 KB, Oracle9*i*AS Web Cache uses two 32 KB buffers to store the HTTP body.
   - Regardless of the size of the body, Oracle9*i*AS Web Cache uses 4 KB to store the HTTP response header.

   Use the following formula to determine an estimate of the maximum memory needed:

   ```
   ( X * ( 4KB + 4KB ) ) + ( Y * (( [m/32] * 32KB ) + 4KB )) + basemem
   ```

   In the formula:

   - *X* is the number of documents smaller than 4 KB.
   - 4KB is the buffer size for the HTTP body for documents smaller than 4 KB.
   - 4KB is the buffer size for the HTTP response header.

- *Y* is number of documents that are 4 KB or larger.
- `[m/32]` is the ceiling of `m` (the average size, in kilobytes, of documents 4 KB or larger) divided by `32`. A **ceiling** is the closest integer that is greater than or equal to the number.
- `32KB` is the buffer size for the HTTP body for documents that are 4 KB or larger.
- *basemem* is the base amount of memory needed by Oracle9*i*AS Web Cache to process requests. This amount includes memory for internal functions such as lookup keys 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 roughly 6 KB to 25 KB of memory. For example, to support 1000 concurrent requests, you need between 6 MB and 25 MB of memory.

  For ESI requests, each concurrent request needs roughly the following amount of memory:

  ```
  60KB + (number of ESI fragments * [6KB to 25KB])
  ```

  For example, for a document with 10 ESI fragments, use the following calculation:

  ```
  60KB + (10 * [6KB to 25KB]) = 120KB to 330KB
  ```

  That is, you need between 120 KB and 330 KB of memory for one 10-fragment document. To support 1000 concurrent requests, you need roughly between 120 MB to 330 MB of memory.

For example, assume that you want to cache 5000 documents that are smaller than 4 KB and 2000 documents that are 4 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 * (4KB + 4KB) ) + ( 2000 * (( [54/32] * 32KB ) + 4KB )) + (500 *[6KB
to 25KB])
```

Using the formula, you need:

- 40,000 KB for the smaller documents.
- 136,000 KB for the larger documents. For the HTTP body, you need 64 KB (two 32 KB buffers) for each document, given the average size of 54 KB. For the HTTP response header, you need 4 KB for each document.
- 3,000 KB to 12,500 KB for the base amount of memory needed to process 500 concurrent requests.

This results in an estimate of 179,000 KB to 188,500 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 your documents are stored in the cache at any given time. That means that you may not need the maximum memory derived from the preceding formula.

---

3. Configure Oracle9*i*AS 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 **Cache-Specific Configuration > 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 Oracle9*i*AS Web Cache Manager main window, click **Apply Changes.**

4. Restart Oracle9*i*AS Web Cache.

5. Using a simulated load or an actual load, monitor the cache to see how much memory it really uses in practice.

   Remember that the cache is empty when Oracle9*i*AS 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 Web Cache Statistics page provides information about the current memory use and the maximum memory use.

To access the Web Cache Statistics page, from the navigator pane, select **Administration > Monitoring > Web Cache Statistics.** Note the following metrics in the Cache Overview table:

- **Size of Documents in Cache** shows the current *logical* size of the cache. The logical size of the cache 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. The physical size of the cache is the amount of data memory allocated by Oracle9*i*AS Web Cache for cache storage and operation. This number is always smaller than the process size shown by operating system statistics because the Oracle9*i*AS 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 90% 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,** Oracle9*i*AS Web Cache begins garbage collection. That is, Oracle9*i*AS 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 **Cache-Specific Configuration > Resource Limits** page to increase the maximum cache size.

Make Sure You Have Sufficient Network Bandwidth

When you use Oracle9*i*AS Web Cache, make sure that each node has sufficient network bandwidth to accommodate the throughput load. Otherwise, the network may be saturated but Oracle9*i*AS Web Cache has additional capacity. For example, if your 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 under utilized 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 Oracle9*i*AS 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. You must 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 you intend to serve concurrently at any given time.
- The average size of a page and the average number of requests per page.
- 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. If a large percentage of requests are cache misses, the requests are forwarded to the application Web server. Those requests consume additional network bandwidth, resulting in longer response times.
- How quickly a page is processed. Use a network monitoring utility, such as `ttcp`, to determine how quickly your system processes a page.

- 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. You set the cluster member capacity using the **Administration > Cluster Configuration** page of Oracle9*i*AS Web Cache Manager.

Use various tools, such as those available with the operating system and with Oracle9*i*AS Web Cache, to help you 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 page is processed. The Oracle9*i*AS 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 Oracle9*i*AS Web Cache Manager, select **Cache-Specific Configuration > 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 Oracle9*i*AS Web Cache Manager main window, click **Apply Changes.**

Do not set the value to an arbitrary high value. Oracle9*i*AS 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 Oracle9*i*AS Web Cache server attempts to reserve the maximum number of file descriptors when it starts. If the `webcachectl` utility can be run as `root`, you can increase this number. For example, on Sun Solaris, you can increase the maximum number of file descriptors by setting the `rlim_fd_max` parameter. If `webcachectl` is not able to run as `root`, the Oracle9*i*AS Web Cache server logs an error message and fails to start.

To run the `webcachectl` utility as the root user, ensure that `root.sh` was run during installation. If `root.sh` was not run during installation, then run it at this time from the `$ORACLE_HOME` directory. See the *Oracle9iAS Web Cache Administration and Deployment Guide* for further details regarding Solaris commands and information about the `webcachectl` utility.

Oracle9*i*AS 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 Oracle9*i*AS Web Cache. You set the maximum number of incoming connections using the **Cache-Specific Configuration > Resource Limits** page of the Oracle9*i*AS 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. You set the capacity using the **Administration > Cluster Configuration** page of the Oracle9*i*AS Web Cache Manager.
- `Total_WS_Capacity` is the sum of the capacity for all configured application Web Servers. You set the capacity using the **General Configuration** > **Application Web Servers** page of Oracle9*i*AS 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 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 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 **Administration > Cluster Configuration** from the navigator pane of Oracle9*i*AS Web Cache Manager.

- `100` is the number of connections reserved for internal use by Oracle9*i*AS Web Cache.

Connections on Windows NT and Windows 2000

On Windows NT and Windows 2000, 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. You can modify the maximum number of ports by editing the Windows registry. Windows NT and Windows 2000 allow up to 65534 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 65534 - 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. (You set the maximum number of incoming connections using the **Cache-Specific Configuration > Resource Limits** page of the Oracle9*i*AS Web Cache Manager. You set the cluster member capacity using the **Administration > Cluster Configuration** page.)

On Windows NT and Windows 2000, Oracle9*i*AS 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.

8

Optimizing PL/SQL Performance

This chapter discusses the techniques for optimizing PL/SQL performance in Oracle9*i* Application Server.

This chapter contains:

PL/SQL Performance in Oracle9*i*AS - Overview

This chapter describes several techniques to improve the performance of your PL/SQL application in Oracle9*i* Application Server (Oracle9*i*AS).

Table 8–1 lists recommendations for Database Access Descriptor (DAD) parameters and settings. By default, these DAD parameters are specified in the file `plsql.conf` in the directory `$ORACLE_HOME/Apache/modplsql/conf`.

Table 8–2 lists caching options.

Table 8–1 Database Access Descriptor (DAD) Parameters

Parameter	Setting
`PlsqlAlwaysDescribeProcedure`	Set this to "Off" for best performance.
`PlsqlDatabaseConnectString`	Use the host:port:sid format instead of a TNS entry
`PlsqlFetchBufferSize`	Default=128 For multi-byte character sets like Japanese, Chinese, setting this to 256 will give better performance
`PlsqlIdleSessionCleanupInterval`	Default=15 (minutes) Increasing this parameter allows a pooled database connection to stay around for longer times
`PlsqlLogEnable`	Default=off This parameter should be set to "Off" unless recommended by Oracle support for debugging purposes
`PlsqlMaxRequestsPerSession`	Default=1000 If the PL/SQL application does not leak resources/memory, this parameter can be tuned higher (for example, 5000)
`PlsqlNLSLanguage`	Setting this parameter to match the database NLS language will disable overheads in character set conversions occurring in Oracle Net Services.
`PlsqlSessionStateManagement`	Set this parameter to "StatelessWithFastResetPackageState" if the database is 8.1.7.2 or above. Oracle9*i*AS Portal is not yet certified with the mode `StatelessWithFastResetPackageState`. For Oracle9*i*AS Portal, set this parameter to the value `StatelessWithResetPackageState`.

Table 8–2 Caching Options

Expires Technique	Best performance - for content that changes predictably See Also: "Using the Expires Technique" on page 8-20
Validation technique	Good performance - for content that changes unpredictably See Also: "Using the Validation Technique" on page 8-16
System-level caching	Improves performance by caching one copy for everyone on system See Also: "System- and User-level Caching with PL/SQL Web Applications" on page 8-23

See Also:

- Appendix A, "Oracle9iAS Performance Metrics" for information on `mod_plsql` metrics
- Chapter 6" Oracle HTTP Server Modules" in the *Oracle HTTP Server Administration Guide* for information on DAD Parameters
- *Oracle9i Application Server mod_plsql User's Guide*
- *Oracle9i Application Server PL/SQL Web Toolkit Reference*

Performance Tuning Issues for mod_plsql

When tuning `mod_plsql` to improve the performance of PL/SQL in 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 connection pooling logic in `mod_plsql` can be best explained with an example. Consider the following typical scenario:

1. The Oracle9*i* Application 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 previous example are:

- 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.
- One Oracle HTTP Server process cannot use or share the connection pool created by another process.

- Each Oracle HTTP Server process pools at most one database connection for each DAD.
- User sessions are switched within a pooled database connection for a DAD. For DADs based on Oracle9*i*AS 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.
- Multiple DADs may point to the same database instance, but database connections are not shared across DADs even within the same process.
- Unused DADs do not result in any database connections.

In the worst case scenario, the total number of database connections that can be 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 Oracle9*i* 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 amount of database sessions.

For example, if there are three Oracle9*i*AS 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 applications to connect.

Because database connections cannot be shared across `httpd` processes, 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. Whenever Oracle HTTP Server becomes threaded in the future, `mod_plsql` will allow for true connection pooling. If the number of database sessions is a concern, then refer to the "Two-Listener Strategy" on page 8-11 for details on how to address this problem.

Closing Pooled Database Sessions

Pooled database sessions are closed under the following circumstances:

- 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 re-established. This is done to make sure that any resource leaks in the PL/SQL application, or in the Oracle client/server side, do not adversely affect the system. This default of 1000 can be changed by tuning the DAD configuration parameter `PlsqlMaxRequestsPerSession`.

- 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 get 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 get cleaned up based on the configuration of `PlsqlMaxRequestsPerSession`. This default of 15 minutes can be changed by tuning the `mod_plsql` configuration parameter `PlsqlIdleSessionCleanupInterval`. Consider increasing the default for better performance in cases where the site is not heavily loaded.

- When the Oracle HTTP Server process goes down

  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 Oracle9*i*AS 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 Administration 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 Oracle9*i*AS 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"

Performance Tuning Areas in mod_plsql

While using `mod_plsql`, there are three areas that affect performance and scalability:

- PL/SQL Application
- Connection Pooling and Oracle HTTP Server Configuration
- Tuning the Number of Database Sessions

PL/SQL Application

PL/SQL Gateway users should consider the following topics when developing PL/SQL applications:

- Database Access Descriptors (DADs)

  You should restrict the number of DADs that are used on each Oracle9*i*AS node. Note that performance is not affected if you have additional DADs that are not being used. See *Oracle9i* Application Server *PL/SQL Web Toolkit Reference* for details.

- 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.

- PL/SQL 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.
- PL/SQL 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`.
- Procedures should make use of the more performant 2-parameter style flexible parameter passing rather than the 4-parameter style parameter passing.

Connection Pooling and Oracle HTTP Server Configuration

- Creating a new database connection is an expensive operation and it is best if every request does not have to open and close a database connection. 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. This reduces the number of database sessions at the expense of performance.
- 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 Oracle9*i*AS 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.

- 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 `mod_plsql` applications, this should not be set to `0`.
- 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 Oracle9*i*AS listener, enabling the `KeepAlive` parameter results in performance improvements. In such cases, make sure to tune the `KeepAliveTimeout` parameter appropriately.
- You may want to lower down 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 may start getting timed out.
- 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 Oracle9*i*AS listener with `mod_expires`. Refer to *Oracle9i* Application Server *mod_plsql User's Guide* for details on how to use `mod_expires`. You should also consider front-ending your Web site with Oracle9*i*AS Web Cache.
  - How do I know if the Web site can benefit from the use of `mod_expires`?
    - Use Netscape to visit a few heavily-accessed Web pages on the site. On each page, right click the mouse and select `View Info` from the pop up menu. 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 Administration Guide* for more details.

  - If you are using Oracle9*i*AS 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=259200'
     </Files>
</Directory>
```

    Refer to the *Oracle9iAS Web Cache Administration and Deployment 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, 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.
  - 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.

Tuning the Number of Database Sessions

- The `processes` parameter 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, maximum number of Oracle HTTP Server processes, and the number of Oracle9*i*AS instances.
- Using a two-listener strategy or using Multi Threaded Server (MTS) reduces the number of database sessions. See "Two-Listener Strategy" on page 8-11.
- 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.
- Front ending your Oracle HTTP Server with Oracle9*i*AS Web Cache reduces the requirement to have a high number of processes for your HTTP configuration, resulting in lesser number of database sessions.

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 Oracle9*i* 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 Oracle9*i*AS listener is running on port 7777 of mylsnr1.mycompany.com. First, you can install another Oracle9*i*AS 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 Oracle9*i*AS listener running on Port 7777, edit ORACLE9IAS_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 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 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 `ORACLE9IAS_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 Oracle9*i*AS listener running on Port 8888. Add the following lines to the `ORACLE9IAS_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 Oracle9*i*AS 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.

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 Oracle9*i* 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 Oracle9*i*AS,

the exclamation mark is now required for flexible parameter passing in Oracle9*i* Application Server. In Oracle9*i*AS, 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 Oracle9*i* 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 Web Applications

Caching can improve the performance of PL/SQL Web applications. 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 application determines the page will be cached, or should be generated again.
- System- and User-level Caching with PL/SQL 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: *Oracle9i Application Server 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 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 an application that serves dynamic content that can change at any given time. In this case, the application needs full control over what is being served. Validation caching always asks the 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 application. `mod_plsql` 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 tag and cache level:

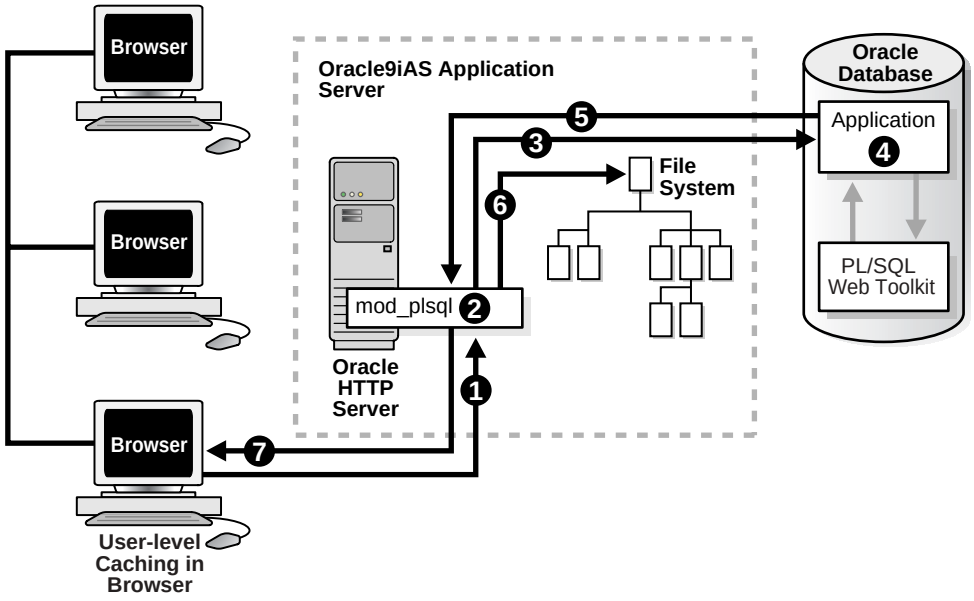
   ```
   owa_cache.set_cache(p_etag, p_level);
   ```

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;
   ```

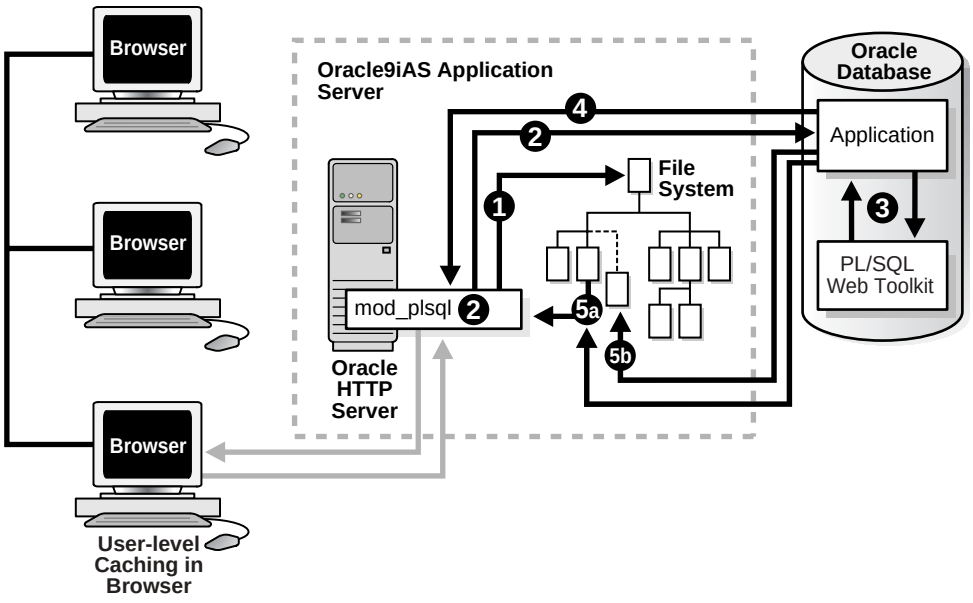
   These owa functions get the tag and caching level.
4. The application sends the caching information to `mod_plsql`.

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 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:

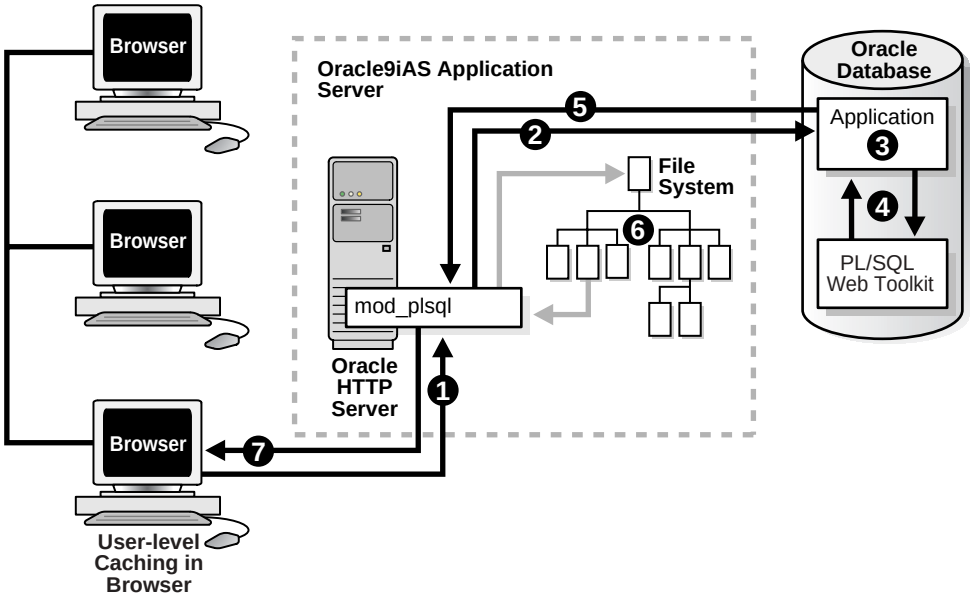
   ```
   owa_cache.set_expires(p_expires, p_level);
   ```

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



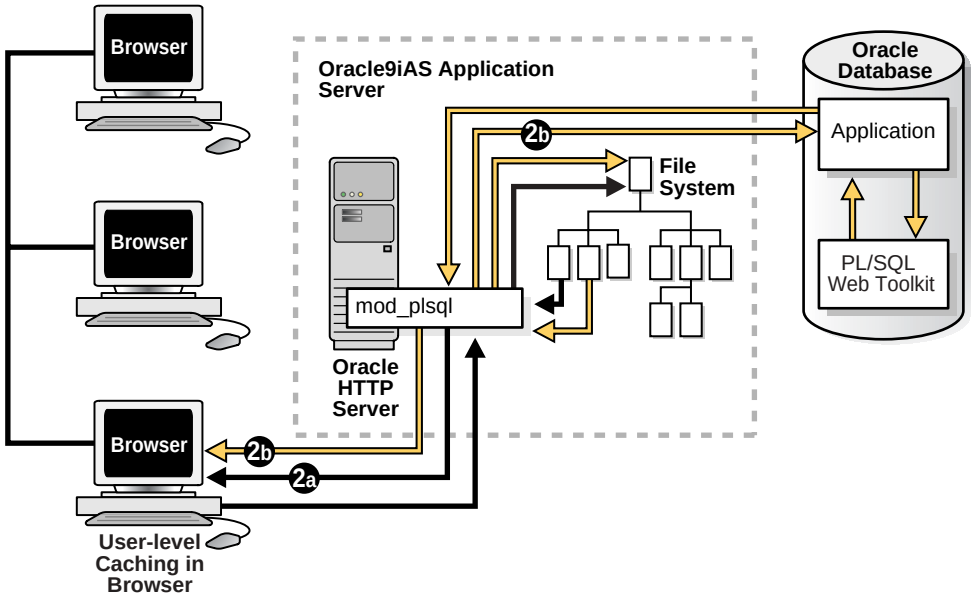
Second Request Using the Expires Technique

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 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 an application that generates content that is intended for everybody using the 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 table below 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 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.

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.

These are 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.

Table 8–6 (Cont.) Primary owa_cache Functions

owa Functions	Purpose
`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.

Other Oracle HTTP Server Directives

Table 8–7 lists some of the Oracle HTTP Server directives that need to be tuned appropriately for your configuration. Adjust these settings for the directives listed in Table 8–7 to values appropriate for your system.

Table 8–7 Default Settings

Directive	Default Value
`KeepAlive`	On
`KeepAliveTimeout`	15 seconds
`MaxClients`	150
`MaxKeepAliveRequests`	100
`MaxRequestsPerChild`	10
`MaxSpareServers`	10
`MinSpareServers`	5
`StartServers`	5

See Also:

- Chapter 5, "Optimizing Oracle HTTP Server"
- Chapter 3, "Managing Server Processes" and Chapter 4, "Managing the Network Connection" in the *Oracle HTTP Server Administration Guide*.

A

Oracle9*i*AS Performance Metrics

This appendix lists metrics that can help you analyze Oracle9*i*AS performance. The metrics fall into several distinct areas, such as Oracle HTTP Server, Oracle9*i*AS Containers for J2EE (OC4J), and Portal. Each table in this chapter lists the metrics that are included in a corresponding DMS metric table.

This appendix contains:

Oracle HTTP Server Metrics

The tables, Table A–1,Table A–2, Table A–3 describe the Oracle HTTP Server metrics.

The metric table name is ohs_server.

Table A–1 HTTP Server Metrics (ohs_server)

Metric	Description	Unit
handle.maxTime	Maximum time spent in module handler	usecs
handle.minTime	Minimum time spent in module handler	usecs
handle.avg	Average time spent in module handler	usecs
handle.active	Chile servers currently in the handle processing phase	threads
handle.time	Total time spent in module handler	usecs
handle.completed	Number of times the handle processing phase has 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.avg	Average time required to service an HTTP request	usecs
request.active	Child servers currently in the request processing phase	threads
request.time	Total time required to service HTTP requests	usecs
request.completed	Number of HTTP request completed	ops
connection.maxTime	Maximum time spent servicing any HTTP connection	usecs
connection.minTime	Minimum time spent servicing any HTTP connection	usecs
connection.avg	Average time spent servicing HTTP connections	usecs
connection.active	Number of connections currently open	threads
connection.time	Total time spent servicing HTTP connections	usecs

Aggregate Module Metrics

Table A–2 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–3 Apache/Modules/mod_*.c Metrics (ohs_module)

Metric	Description	Unit
decline.count	Number of requests declined	ops
handle.maxTime	Maximum time required for this module	usecs
handle.minTime	Minimum time required for this module	usecs
handle.avg	Average time required for this module	usecs
handle.active	Number of requests currently being handled by this module	requests
handle.time	Total time required for this module	usecs
handle.completed	Number of requests handled by this module	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–4 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
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 JDBC metrics collected in Oracle9*i*AS.

JDBC Driver Metrics

There is one set of JDBC Driver metrics per JVM. The metric table name is `JDBC_Driver`.

Table A–5 /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–6 /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
`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

Table A–6 (Cont.) /JDBC/data-source-name - JDBC_Data Source Metrics

Metric	Description	Unit
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

The metric table name is JDBC_Connection. There is one set of JDBC Connection metrics per connection.

Table A–7 /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.	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

The metric table name is JDBC_Connection. There is one set of JDBC data source specific connection metrics per data source per connection.

Table A–8 /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.	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

The metric table name is `JDBC_Statement`.

There is a set of statement metrics per connection per statement.

> **Note:** The JDBC statement metrics are only available when JDBC statement caching is enabled.

Table A–9 /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.	msecs
`Fetch.time`	The time this statement has spent in other fetches.	msecs
`SQLText.value`	The SQL being executed.	

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 when JDBC statement caching is enabled.

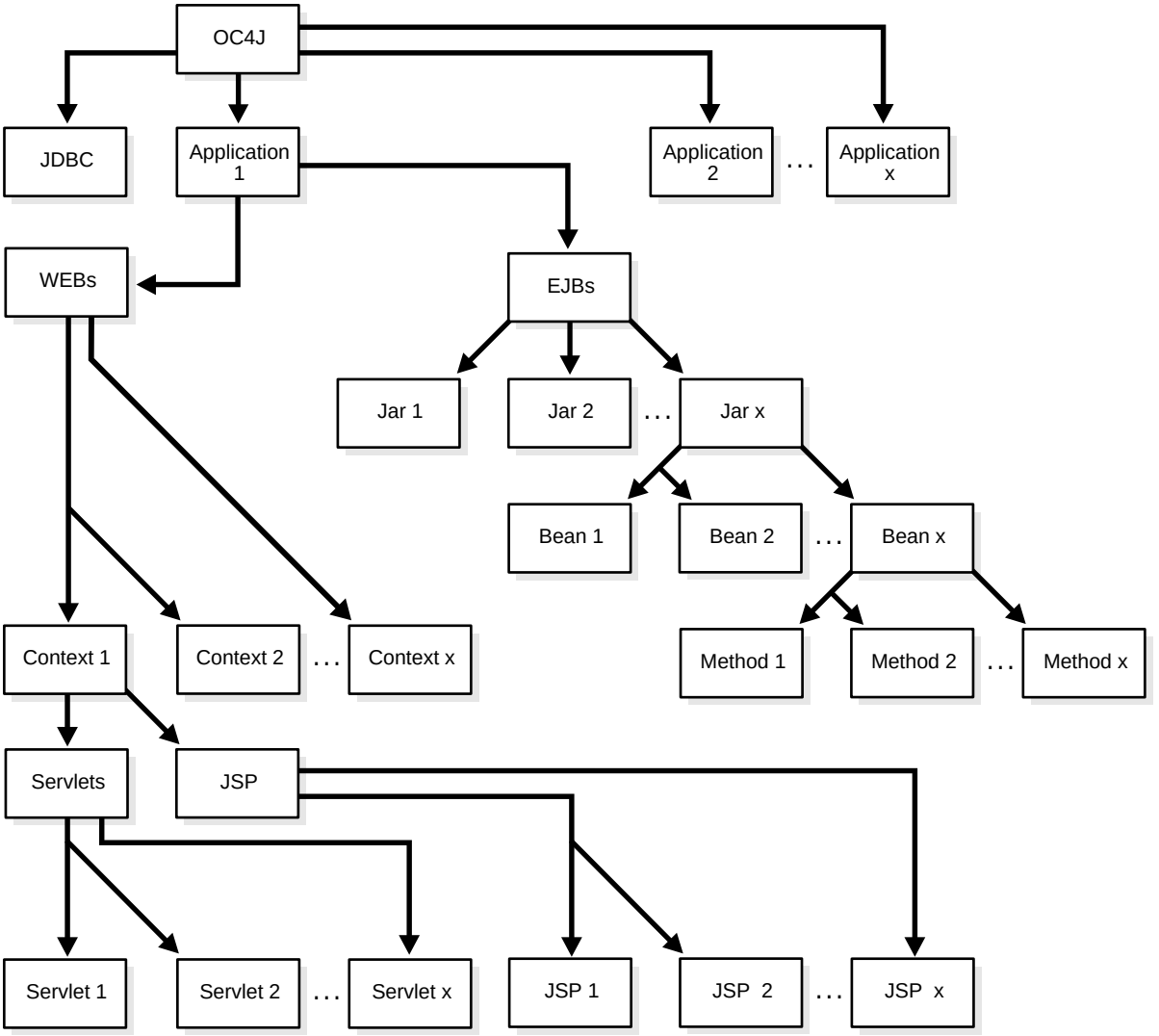
Table A–10 /JDBC/data-source-name/CONNECTION/STATEMENT JDBC Statement Metrics

Metric	Description	Unit
`Execute.time`	The time this statement has spent executing the SQL including the first fetch.	msecs
`Fetch.time`	The time this statement has spent in other fetches.	msecs
`SQLText.value`	The SQL being executed.	

J2EE Application Metrics - OC4J Metrics

Figure A–1 illustrates the metrics in a J2EE application.

Figure A–1 Structure of Performance Metrics for a J2EE Application



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–11 OC4J/application/WEBs Metrics

Metric	Description	Unit
`processRequest.time`	Total time spent servicing this application's web requests	msecs
`processRequest.completed`	Number of web requests processed by this application	ops
`processRequest.minTime`	Minimum time spent servicing a web request	msecs
`processRequest.maxTime`	Maximum time spent servicing a web request	msecs
`processRequest.avg`	Average time spent servicing web requests	msecs
`processRequest.active`	Current number of threads servicing web requests	
`resolveContext.time`	Total time spent to create / find the servlet context. Each web module (WAR) maps to a servlet context	msecs
`resolveContext.completed`	Count of completed context resolves	ops
`resolveContext.minTime`	Minimum time spent to create / find the servlet context	msecs
`resolveContext.maxTime`	Maximum time spent to create / find the servlet context	msecs
`resolveContext.avg`	Average time spent to create / find the servlet context	msecs
`resolveContext.active`	Current number of threads trying to create / find the servlet context	
`parseRequest.time`	Total time spent to read / parse requests from the socket	msecs
`parseRequest.completed`	Number of web requests that have been parsed	ops
`parseRequest.minTime`	Minimum time spent to read / parse requests	msecs
`parseRequest.maxTime`	Maximum time spent to read / parse requests	msecs
`parseRequest.avg`	Average time spent to read / parse requests	msecs
`parseRequest.active`	Current number of threads trying to read / parse AJP or HTTP requests	

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–12 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
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

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–13 OC4J/application/WEBs/context /SERVLETS/servlet Metrics

Metric	Description	Unit
service.time	Total time spent on the servlet's service() call	msecs
service.completed	Total number of calls to service()	
service.minTime	Minimum time spent on a servlet's service() call	msecs

Table A–13 (Cont.) OC4J/application/WEBs/context /SERVLETS/servlet Metrics

Metric	Description	Unit
service.maxTime	Maximum time spent on a servlet's service() call	ops
service.avg	Average time spent in servicing the servlet	msecs
service.active	Current number of threads servicing this servlet	msecs

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–14 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	

JSP Metrics

There is one set of metrics for each JSP in each Web module. the data for availableInstance.* appears only for non-threadsafe JSPs.

The metric table name is oc4j_jsp.

Table A–15 OC4J/application/WEBs/context /JSPjsp_name Metrics

Metric	Description	Unit
service.time	Time to serve a JSP (that is, actual execution time of the JSP)	msecs
service.completed	Number of requests for JSPs processed by this JSP	ops
service.minTime	Minimum time spent servicing the JSP	msecs
service.maxTime	Maximum time spent servicing the JSP	msecs
service.avg	Average time spent servicing the JSP	msecs

Table A–15 (Cont.) OC4J/application/WEBs/context /JSPjsp_name Metrics

Metric	Description	Unit
service.active	Current number of active requests for the JSP	
availableInstances.value	Number of available (that is, created) instances. Only used when threadsafe=false	instances
activeInstances.value	Number of active instances. Only used when threadsafe=false	instances

EJB Metrics

EJB Bean Metrics

Oracle9*i*AS 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–16 OC4J/application/EJBs/ejb-jar-module/ejb-name Metrics

Metric	Description	Unit
transaction-type.value	Transaction type. Possible values: container or bean	
session-type.value	Session type. Possible values: stateful or stateless	
bean-type.value	Bean type Possible values: session or entity bean	
exclusive-write-access.value	Exclusive write access value Possible values: true or false	
isolation.value	Isolation value. Possible values: serializable, uncommitted, committed, repeatable_read, none	
persistence-type.value	Persistence type: Possible values: bean or entity bean	

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 *Oracle9iAS Containers for J2EE Enterprise JavaBeans Developer's Guide and Reference* for information on automatically generated wrappers.

Table A–17 OC4J/application/EJBs/ejb-jar-module/ejb-name/method-name Metrics

Metric	Description	Unit
`client.time`	Time spent inside the actual implementation of this method	msecs
`client.completed`	Number of requests for beans processed by this application	ops
`client.minTime`	Minimum time spent inside the actual implementation of this method	msecs
`client.maxTime`	Maximum time spent inside the actual implementation of this method	msecs
`client.avg`	Average time spent inside the actual implementation of this method	msecs
`client.active`	Current number of threads accessing the actual implementation of this method	
`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
`wrapper.completed`	Number of requests for beans processed by this application	ops
`wrapper.minTime`	Minimum time spent inside the automatically generated wrapper method	msecs
`wrapper.maxTime`	Maximum time spent inside the automatically generated wrapper method	msecs
`wrapper.avg`	Average time spent inside the automatically generated wrapper method	msecs
`wrapper.active`	Current number of threads accessing the automatically generated wrapper method	
`trans-attribute.value`	Transaction attribute. Possible values: `NotSupported`, `Supports`, `RequiresNew`, `Mandatory`, and `Never`	

Table A–17 (Cont.) OC4J/application/EJBs/ejb-jar-module/ejb-name/method-name Metrics

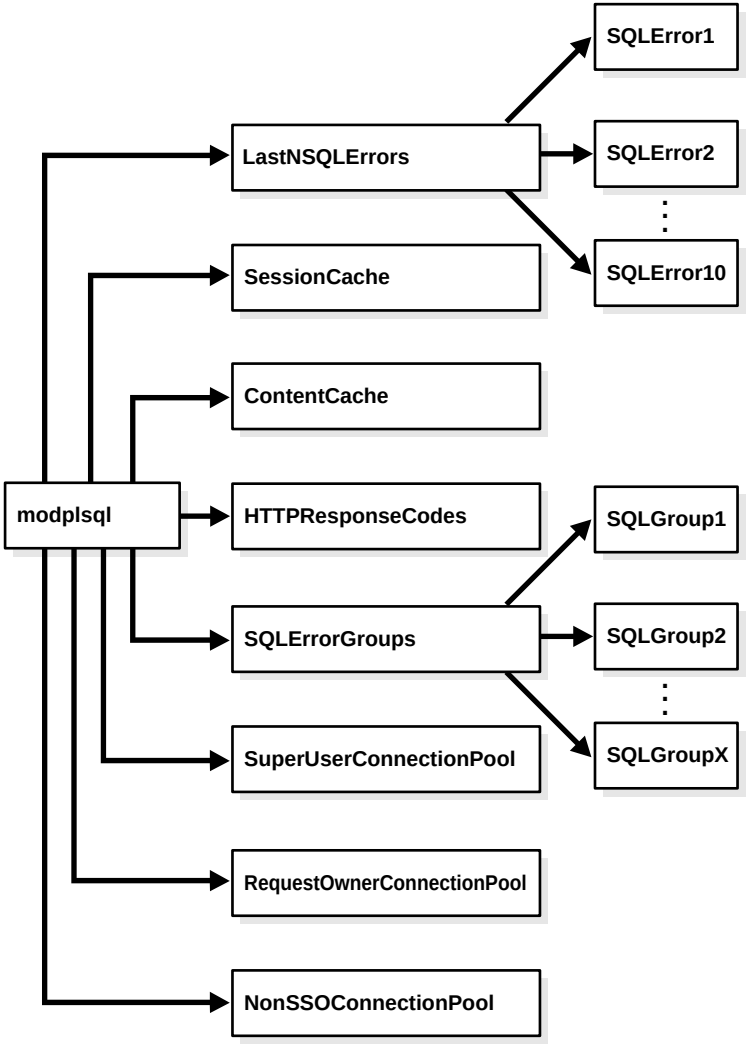
Metric	Description	Unit
ejbPostCreate.time	Time spent in the ejbPostCreate method (entity beans)	msecs
ejbPostCreate.completed	Number of times this ejbPostCreate has been called	ops
ejbPostCreate.minTime	Minimum time spent in ejbPostCreate	msecs
ejbPostCreate.maxTime	Maximum time spent in ejbPostCreate	msecs
ejbPostCreate.avg	Average time spent in ejbPostCreate	msecs
ejbPostCreate.active	Current number of threads executing ejbPostCreate	

Portal Metrics

This section describes the Oracle9*i*AS Portal metrics.

Figure A–2, "mod_plsql Metric Tree" shows the structure of the `mod_plsql` metrics. The tables in this section describe the relevant metrics.

Figure A–2 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–18 lists the set of metrics for the mod_plsql session cache.

The metric table name is modplsql_Cache.

Table A–18 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–19 lists the set of metrics for the mod_plsql content cache.

The metric table name is modplsql_ContentCache.

Table A–19 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–20 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–20 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–21 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–21 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–22 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–22 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–23 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–23 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–24 lists the set of metrics for the super user connection pool.

The metric table name is `modplsql_DatabaseConnectionPool`:

```
/modplsql/SuperUserConnectionPool [type=modplsql_DatabaseConnectionPool]
```

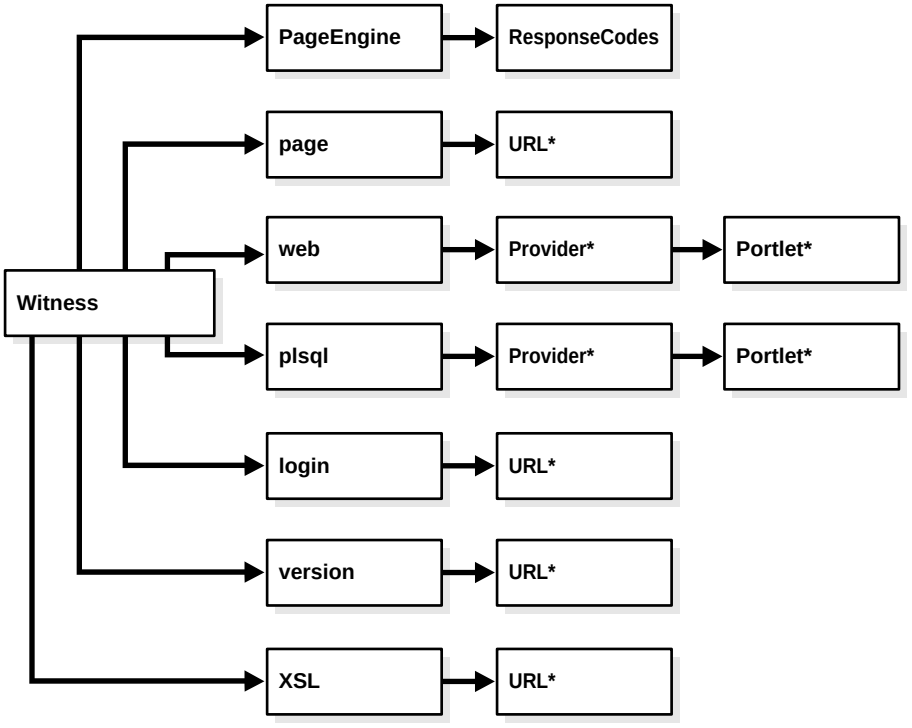
Table A–24 mod_plsql/SuperUserConnectionPool Metrics

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

Parallel Page Engine Metrics

Figure A–3, "Parallel Page Engine Metric Tree" shows the structure of the Parallel Page Engine metrics. The tables in this section describe the relevant metrics.

Figure A–3 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–25 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–25 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 (i.e. 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 (i.e. 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

Table A–25 (Cont.) Witness/PageEngine Metrics

Metric	Description	Unit
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
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–26 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–26 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–27 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–27 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–28 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–28 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–29 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–29 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–30 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–30 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–30 (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–31 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–30 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–31 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–32 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–32 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–33 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–32 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–33 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

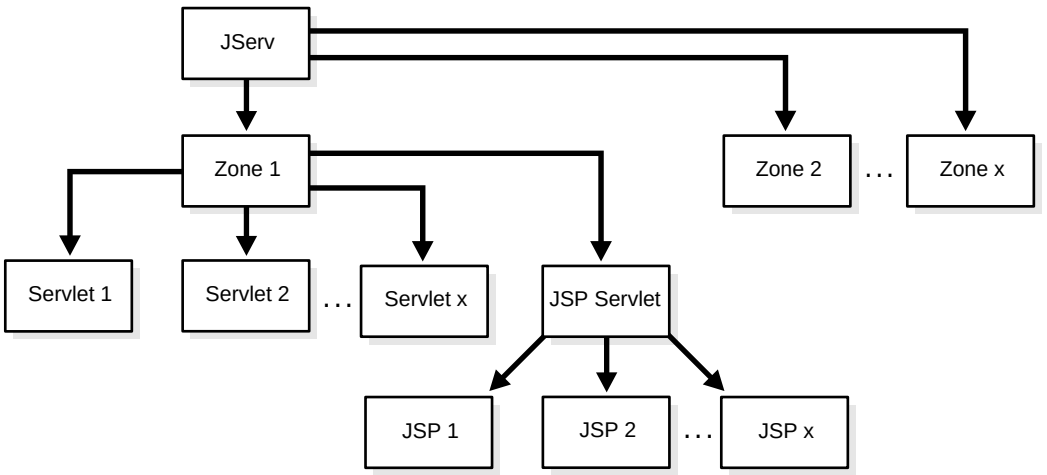
Table A–33 (Cont.) Witness/Web/dad-provider/portlet Metrics

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–4 shows the structure of the JServ metrics. The following tables describe the relevant metrics.

Figure A–4 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–34 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–35 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

Table A–35 (Cont.) jserv/zone Metrics

Metric	Description	Unit
activeSessions.value	The number of sessions which exist in this zone	sessions
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–36 /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

Table A–36 (Cont.) /jserv/zone/servlet Metrics

Metric	Description	Unit
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	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–37 /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	

Table A–37 (Cont.) /jserv/zone/servlet Metrics

Metric	Description	Unit
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	
activeSTMInstances.maxValue	Maximum number of instances concurrently servicing requests for this SingleThreadModel	
activeSTMInstances.value	Total number of instances available for this SingleThreadModel servlet	instances

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W