Oracle® Fusion Middleware

Tuning Performance Guide

12c (12.1.2)

E28643-01

July 2013

Describes how to monitor and optimize performance, configure components for optimal performance, and write highly performant applications in the Oracle Fusion Middleware environment.

Oracle Fusion Middleware Tuning Performance Guide 12c (12.1.2)

E28643-01

Primary Author: Lisa Jamen

Contents

Preface

This guide describes how to monitor and optimize performance, review the key components that impact performance, use multiple components for optimal performance, and design applications for performance in the Oracle Fusion Middleware environment.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Conventions

Audience

Tuning Performance is aimed at a target audience of Application developers, Oracle Fusion Middleware administrators, database administrators, and Web masters.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.`

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Part I

Introduction

This part describes basic performance concepts, how to measure performance, and designing applications for performance and scalability. It contains the following chapters:

1 Introduction and Roadmap

This section describes the contents and organization of this guide.

- Section 1.1, "Document Scope and Audience"
- Section 1.2, "Guide to this Document"
- Section 1.3, "Related Documentation"

1.1 Document Scope and Audience

Tuning Performance is for a target audience of Application developers, Oracle Fusion Middleware administrators, database administrators, and Web masters. This Guide assumes knowledge of Fusion Middleware Administration and hardware performance tuning fundamentals, WebLogic Server, XML, and the Java programming language.

1.2 Guide to this Document

- This chapter, Chapter 1, "Introduction and Roadmap," introduces the objectives and organization of this guide.
- Chapter 2, "Top Performance Areas," describes top tuning areas for Oracle Fusion Middleware and serves as a ’quick start’ for tuning applications.
- Chapter 3, "Performance Planning," describes the performance planning methodology and tuning concepts for Oracle Fusion Middleware.
- Chapter 4, "Monitoring Oracle Fusion Middleware," describes how to monitor Oracle Fusion Middleware and its components to obtain performance data that can assist you in tuning the system and debugging applications with performance problems.
- Chapter 5, "Using the Oracle Dynamic Monitoring Service" provides an overview and features available in the Oracle Dynamic Monitoring Service (DMS).
- Chapter 6, "Oracle HTTP Server Performance Tuning," discusses the techniques for optimizing Oracle HTTP Server performance, the Web server component for Oracle Fusion Middleware. It provides a listener for Oracle WebLogic Server and the framework for hosting static pages, dynamic pages, and applications over the Web.
- Chapter 7, "Oracle Metadata Service (MDS) Performance Tuning," provides tuning tips for Oracle Metadata Service (MDS). MDS is used by Oracle Application Development Framework to manage metadata.
- Chapter 8, "Oracle Application Development Framework Performance Tuning," provides basic guidelines on how to maximize the performance and scalability of

the ADF stack in applications. Oracle ADF is an end-to-end application framework that builds on Java Platform, Enterprise Edition (Java EE) standards and open-source technologies to simplify and accelerate implementing service-oriented applications. This chapter covers design time, configuration time, and deployment time performance considerations.

- Chapter 9, "Oracle TopLink (EclipseLink) JPA Performance Tuning," provides some of the available performance options for Java Persistence API (JPA) entity architecture. Oracle TopLink includes EclipseLink as the JPA implementation.

1.3 Related Documentation

For more information, see the following documents in the Oracle Fusion Middleware 12c (12.1.2) documentation set:

- *Administering Oracle Fusion Middleware*
- *Understanding Oracle Fusion Middleware*
- *Securing Applications with Oracle Platform Security Services*
- *High Availability Guide*
- *Tuning Performance of Oracle WebLogic Server*
- *Administering Oracle HTTP Server*
- *Administering Web Services*

2 Top Performance Areas

This chapter describes the top tuning areas for Oracle Fusion Middleware. It covers critical Oracle Fusion Middleware performance areas and provides a quick start for tuning Java applications in the following sections:

2.1 Identifying Top Performance Areas

One of the most challenging aspects of performance tuning is knowing where to begin. This chapter serves as a 'quick start' guide to performance tuning your Oracle Fusion Middleware applications.

Table 2–1 provides a list of common performance considerations for Oracle Fusion Middleware. While the list is a useful tool in starting your performance tuning, it is not meant to be comprehensive list of areas to tune. You must monitor and track specific performance issues within your application to understand where tuning can improve performance. See Chapter 4, "Monitoring Oracle Fusion Middleware" for more information.

Table 2–1 Top Performance Areas for Oracle Fusion Middleware

Performance Area	Description and Reference
Hardware Resources	Ensure that your hardware resources meet or exceed the application's resource requirements to maximize performance. See Section 2.2, "Securing Sufficient Hardware Resources" for information on how to determine if your hardware resources are sufficient.
Operating System	Each operating system has native tools and utilities that can be useful for monitoring purposes. See Section 2.3, "Tuning the Operating System"
Java Virtual Machines (JVMs)	This section discusses best practices and provides practical tips to tune the JVM and improve the performance of a Java EE application. It also discusses heap size and JVM garbage collection options. See Section 2.4, "Tuning Java Virtual Machines (JVMs)".
Database	For applications that access a database, ensure that your database is properly configured to support your application's requirements. See Section 2.6, "Tuning Database Parameters" for more information on garbage collection.
WebLogic Server	If your Oracle Fusion Middleware applications are using the WebLogic Server, see Section 2.5, "Tuning the WebLogic Server".
Database Connections	Pooling the connections so they are reused is an important tuning consideration. See Section 2.7, "Reusing Database Connections"
Data Source Statement Caching	For applications that use a database, you can lower the performance impact of repeated statement parsing and creation by configuring statement caching properly. See Section 2.8, "Enabling Data Source Statement Caching"
Oracle HTTP Server	Tune the Oracle HTTP Server directives to set the level of concurrency by specifying the number of HTTP connections. See Section 2.9, "Controlling Concurrency".
Concurrency	This section discusses ways to control concurrency with Oracle Fusion Middleware components. See Section 2.9, "Controlling Concurrency"
Logging Levels	Logging levels are thresholds that a system administrator sets to control how much information is logged. Performance can be impacted by the amount of information that applications log therefore it is important to set the logging levels appropriately. See Section 2.10, "Setting Logging Levels".

2.2 Securing Sufficient Hardware Resources

A key component of managing the performance of Oracle Fusion Middleware applications is to ensure that there are sufficient CPU, memory, and network resources to support the user and application requirements for your installation.

No matter how well you tune your applications, if you do not have the appropriate hardware resources, your applications cannot reach optimal performance levels. Oracle Fusion Middleware has minimum hardware requirements for its applications and database tier. For details on Oracle Fusion Middleware supported configurations, see "System Requirements and Prerequisites" in *Planning an Installation of Oracle Fusion Middleware*.

Sufficient hardware resources should meet or exceed the acceptable response times and throughputs for applications without becoming saturated. To verify that you have sufficient hardware resources, you should monitor resource utilization over an extended period to determine if (or when) you have occasional peaks of usage or whether a resource is consistently saturated. For more information on monitoring, see Chapter 4, "Monitoring Oracle Fusion Middleware".

> **Tip:** Your target CPU usage should never reach 100% utilization. You should determine a target CPU utilization based on your application needs, including CPU cycles for peak usage.
>
> If your CPU utilization is optimized at 100% during normal load hours, you have no capacity to handle a peak load. In applications that are latency sensitive and maintaining a fast response time is important, high CPU usage (approaching 100% utilization) can increase response times while throughput stays constant or even decreases. For such applications, a 70% - 80% CPU utilization is recommended. A good target for non-latency sensitive applications is about 90%.

If any of the hardware resources are saturated (consistently at or near 100% utilization), one or more of the following conditions may exist:

- The hardware resources are insufficient to run the application.
- The system is not properly configured.
- The application or database must be tuned.

For a consistently saturated resource, the solutions are to reduce load or increase resources. For peak traffic periods when the increased response time is not acceptable, consider increasing resources or determine if there is traffic that can be rescheduled to reduce the peak load, such as scheduling batch or background operations during slower periods.

Oracle Fusion Middleware provides a variety of mechanisms to help you control resource concurrency; this can limit the impact of bursts of traffic. However, for a consistently saturated system, these mechanisms should be viewed as temporary solutions. For more information see Section 2.9, "Controlling Concurrency".

2.3 Tuning the Operating System

Each operating system has native tools and utilities that can be useful for monitoring and tuning purposes. Native operating system commands enable you to monitor CPU utilization, paging activity, swapping, and other system activity information.

For details on operating system commands, and guidelines for performance tuning of the network or operating system, refer to the documentation provided by the operating system vendor.

2.4 Tuning Java Virtual Machines (JVMs)

How you tune your Java virtual machine (JVM) greatly affects the performance of Oracle Fusion Middleware and your applications. For more information on tuning your JVM, see "Tuning Java Virtual Machines (JVM)" in *Tuning Performance of Oracle WebLogic Server*.

2.5 Tuning the WebLogic Server

If your Oracle Fusion Middleware applications are using the WebLogic Server, see "Tuning WebLogic Server" in *Tuning Performance of Oracle WebLogic Server*.

2.6 Tuning Database Parameters

To achieve optimal performance for applications that use the Oracle database, the database tables you access must be designed with performance in mind. Monitoring and tuning the database ensures that you get the best performance from your applications.

Note: The information in this section is a subset of database tuning information for Fusion Middleware. More information can be found in *Oracle Database Performance Tuning Guide*. Make sure that you have also reviewed your database tuning documentation.

This section covers the following:

- Tuning Database Parameters
- Tuning Redo Logs Location and Sizing
- Tuning Automatic Segment-Space Management (ASSM)

Note: Always review the tuning guidelines in your database-specific vendor documentation. For more information on tuning the Oracle database, see the *Oracle Database Performance Tuning Guide*.

2.6.1 Tuning Database Parameters

The following tables provide common **init.ora** parameters and their descriptions. Consider following these guidelines to set the database parameters. Ultimately, however, the DBA should monitor the database health and tune parameters based on the need. See Table 2–2 for more information:

Table 2–2 Important inti.ora Oracle 11g Database Tuning Parameters

Database Parameter	Description
AUDIT_TRAIL	If there is NO policy to audit db activity, consider setting this parameter to NONE. Enabling auditing can impact performance.
MEMORY_MAX_TARGET	MEMORY_MAX_TARGET specifies the maximum value to which a DBA can set the MEMORY_TARGET initialization parameter.
MEMORY_TARGET	Consider setting the MEMORY_TARGET to NONE. Set SGA and PGA separately as setting MEMORY_TARGET does not allocate sufficient memory to SGA and PGA as needed.

Table 2–2 (Cont.) Important inti.ora Oracle 11g Database Tuning Parameters

Database Parameter	Description
PGA_AGGREGATE_ TARGET	Consider using a value of 1G for PGA initially and then monitor the production database on daily basis and adjust SGA and PGA accordingly. If the database server has more memory, consider setting PGA_AGGREGATE_ TARGET to a value higher than 1G based on usage needs.
SGA_MAX_SIZE	Consider setting MEMORY_TARGET instead of setting SGA and the PGA separately.
SGA_TARGET	Consider using a value of 2G for SGA is 2G to start with and initially and then monitor the production database on daily basis and adjust SGA and PGA accordingly. If the database server has more memory, consider setting SGA_TARGET to a value higher than 2G based on usage needs.

2.6.2 Tuning Redo Logs Location and Sizing

Tuning the redo log options can provide performance improvement for applications running in an Oracle Fusion Middleware environment, and in some cases, you can significantly improve I/O throughput by moving the redo logs to a separate disk.

Consider having at least 3 redo log groups with 2G of size each. Redo log files should be placed on a disk separate from data files to improve I/O performance.

2.6.3 Tuning Automatic Segment-Space Management (ASSM)

For permanent tablespaces, consider using automatic segment-space management. Such tablespaces, often referred to as bitmap tablespaces, are locally managed tablespaces with bitmap segment space management.

For backward compatibility, the default local tablespace segment-space management mode is `MANUAL`.

While configuring tablespaces, consider setting the extent allocation type to SYSTEM. If the allocation type is set to UNIFORM, it might impact performance.

For more information, see "Free Space Management" in *Oracle Database Concepts*, and "Specifying Segment Space Management in Locally Managed Tablespaces" in *Oracle Database Administrator's Guide*.

2.7 Reusing Database Connections

Creating a database connection is a relatively resource intensive process in any environment. Typically, a connection pool starts with a small number of connections. As client demand for more connections grow, there may not be enough in the pool to satisfy the requests. WebLogic Server creates additional connections and adds them to the pool until the maximum pool size is reached.

One way to avoid connection creation delays is to initialize all connections at server startup, rather than on-demand as clients need them. This may be appropriate if your load is predictable and even. Set the initial number of connections equal to the maximum number of connections in the Connection Pool tab of your data source configuration. Determine the optimal value for the Maximum Capacity as part of your pre-production performance testing.

If your load is uneven, and has a much higher number of connections at peak load than at typical load, consider setting the initial number of connections equal to your typical load. In addition, consider setting the maximum number of connections based on your supported peak load. With these configurations, WebLogic server can free up some connections when they are not used for a period of time.

For more information, see "Tuning Data Source Connection Pool Options" in *Administering JDBC Data Sources for Oracle WebLogic Server*.

2.8 Enabling Data Source Statement Caching

When you use a prepared statement or callable statement in an application or EJB, there may be a performance impact associated with the processing of the communication between the application server and the database server and on the database server. To minimize the processing impact, enable the data source to cache prepared and callable statements used in your applications. When an application or EJB calls any of the statements stored in the cache, the server reuses the statement stored in the cache. Reusing prepared and callable statements reduces CPU usage on the database server, improving performance for the current statement and leaving CPU cycles for other tasks.

Each connection in a data source has its own individual cache of prepared and callable statements used on the connection. However, you configure statement cache options per data source. That is, the statement cache for each connection in a data source uses the statement cache options specified for the data source, but each connection caches it's own statements. Statement cache configuration options include:

- Statement Cache Type—The algorithm that determines which statements to store in the statement cache.
- Statement Cache Size—The number of statements to store in the cache for each connection. The default value is 10. You should analyze your database's statement parse metrics to size the statement cache sufficiently for the number of statements you have in your application.

You can use the Administration Console to set statement cache options for a data source. See "Configure the statement cache for a JDBC data source" in the *Oracle WebLogic Server Administration Console Online Help*.

For more information on using statement caching, see "Increasing Performance with the Statement Cache" in the *Administering JDBC Data Sources for Oracle WebLogic Server*.

2.9 Controlling Concurrency

Limiting concurrency, at multiple layers of the system to match specific usage needs, can greatly improve performance. This section discusses a few of the areas within Oracle Fusion Middleware where concurrency can be controlled.

When system capacity is reached, and a web server or application server continues to accept requests, application performance and stability can deteriorate. There are several places within Oracle Fusion Middleware where you can throttle the requests to avoid overloading the mid-tier or database tier systems and tune for best performance.

- Setting Server Connection Limits
- Configuring Connection Pools
- Tuning the WebLogic Sever Thread Pool

2.9.1 Setting Server Connection Limits

Oracle HTTP Server uses directives in `httpd.conf`. This configuration file specifies the maximum number of HTTP requests that can be processed simultaneously, logging details, and certain limits and time outs.

For more information on modifying the httpd.conf file, see "Configuring Oracle HTTP Server" in *Administering Oracle HTTP Server*.

You can use the `MaxClients` and `ThreadsPerChild` directives to limit incoming requests to WebLogic instances from the Oracle HTTP Server based on your expected client load and system resources. The following sections describe some Oracle HTTP Server tuning parameters related to connection limits that you typically need to tune based on your expected client load. See Chapter 6, "Oracle HTTP Server Performance Tuning" for more information and a more complete list of tunable parameters.

2.9.1.1 MaxClients/ThreadsPerChild

Note: The `MaxClients` parameter is applicable only to UNIX platforms and on Microsoft Windows (mpm_winnt), the same is achieved through the `ThreadsPerChild` and `ThreadLimit` parameters.

The `MaxClients` property specifies a limit on the total number of server threads running, that is, a limit on the number of clients who can simultaneously connect. If the number of client connections reaches this limit, then subsequent requests are queued in the TCP/IP system up to the limit specified (in the ListenBackLog directive).

You can configure the `MaxClients` directive in the httpd.conf file up to a maximum of 8K (the default value is 150). If your system is not resource-saturated and you have a user population of more than 150 concurrent HTTP connections, you can improve your performance by increasing `MaxClients` to increase server concurrency. Increase `MaxClients` until your system becomes fully utilized (85% is a good threshold).

When system resources are saturated, increasing `MaxClients` does not improve performance. In this case, the `MaxClients` value could be reduced as a throttle on the number of concurrent requests on the server.

If the server handles persistent connections, then it may require sufficient concurrent httpd server processes to handle both active and idle connections. When you specify `MaxClients` to act as a throttle for system concurrency, you need to consider that persistent idle httpd connections also consume httpd processes. Specifically, the number of connections includes the currently active persistent and non-persistent connections and the idle persistent connections. When there are no httpd server threads available, connection requests are queued in the TCP/IP system until a thread becomes available, and eventually clients terminate connections.

You can define a number of server processes and the threads per process (`ThreadsPerChild`) to handle the incoming connections to Oracle HTTP Server. The `ThreadsPerChild` property specifies the upper limit on the number of threads that can be created under a server (child) process.

Note: `ThreadsPerChild`, `StartServers`, and `ServerLimit` properties are inter-related with the `MaxClients` setting. All of these properties must be set appropriately to achieve the number of connections as specified by `MaxClients`. See Table 6–1, " Oracle HTTP Server Configuration Properties" for a description of all the HTTP configuration properties.

2.9.1.2 KeepAlive

A persistent, `KeepAlive`, HTTP connection consumes an httpd child process, or thread, for the duration of the connection, even if no requests are currently being processed for the connection.

If you have sufficient capacity, `KeepAlive` should be enabled; using persistent connections improves performance and prevents wasting CPU resources re-establishing HTTP connections. Normally, you should not need to change `KeepAlive` parameters.

Note: The default maximum requests for a persistent connection is 100, as specified with the `MaxKeepAliveRequests` directive in httpd.conf. By default, the server waits for 15 seconds between requests from a client before closing a connection, as specified with the `KeepAliveTimeout` directive in httpd.conf.

2.9.1.3 Tuning HTTP Server Modules

The Oracle HTTP Server (OHS) uses the mod_wl_ohs module to route requests to the underlying Weblogic server or the Weblogic Server cluster. The configuration details for mod_wl_ohs are available in the `mod_wl_ohs.conf` file in the config directory.

For more information on the tuning parameters for mod_wl_ohs see, "Understanding Oracle HTTP Server Modules" in *Administering Oracle HTTP Server*.

2.9.2 Configuring Connection Pools

Connection pooling is configured and maintained per Java runtime. Connections are not shared across different runtimes. To use connection pooling, no configuration is required. Configuration is necessary only if you want to customize how pooling is done, such as to control the size of the pools and which types of connections are pooled.

You configure connection pooling by using a number of system properties at program startup time. Note that these are system properties, not environment properties and that they affect all connection pooling requests.

For applications that use a database, performance can improve when the connection pool associated with a data source limits the number of connections. You can use the `MaxCapacity` attribute to limit the database requests from Oracle Application Server so that incoming requests do not saturate the database, or to limit the database requests so that the database access does not overload the Oracle Application Server-tier resource.

The connection pool `MaxCapacity` attribute specifies the maximum number of connections that a connection pool allows. By default, the value of `MaxCapacity` is set to 15. For best performance, you should specify a value for `MaxCapacity` that matches the number appropriate to your database performance characteristics.

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 `MaxCapacity` option, as specified in all the applications that access the database.

See Also: "JDBC Data Source: Configuration: Connection Pool" in the *Oracle WebLogic Server Administration Console Online Help*.

"Tuning Data Source Connection Pool Options" in *Administering JDBC Data Sources for Oracle WebLogic Server*.

2.9.3 Tuning the WebLogic Sever Thread Pool

By default WebLogic Server uses a single thread pool, in which all types of work are executed. WebLogic Server uses Work Managers to prioritize work based on rules you can define, and run-time metrics, including the actual time it takes to execute a request and the rate at which requests are entering and leaving the pool. There is a default work manager that manages the common thread pool.

The common thread pool changes its size automatically to maximize throughput. WebLogic Server monitors throughput over time and based on history, determines whether to adjust the thread count. For example, if historical throughput statistics indicate that a higher thread count increased throughput, WebLogic increases the thread count. Similarly, if statistics indicate that fewer threads did not reduce throughput, WebLogic decreases the thread count.

Since the WebLogic Server thread pool by default is sized automatically, in most situations you do not need to tune this. However, for special requirements, an administrator can configure custom Work Managers to manage the thread pool at a more granular level for sets of requests that have similar performance, availability, or reliability requirements. With custom work managers, you can define priorities and guidelines for how to assign pending work (including specifying a min threads or max threads constraint, or a constraint on the total number of requests that can be queued or executing before WebLogic Server begins rejecting requests).

Use the following guidelines to help you determine when to use Work Managers to customize thread management:

- The default fair share is not sufficient.

  This usually occurs in situations where one application needs to be given a higher priority over another.
- A response time goal is required.
- A minimum thread constraint needs to be specified to avoid server deadlock.
- You use MDBs in your application.

  To ensure MDBs use a well-defined share of server thread resources, and to tune MDB concurrency, most MDBs should be modified to reference a custom work manager that has a max-threads-constraint. In general, a custom work manager is useful when you have multiple MDB deployments, or if you determine that a particular MDB needs more threads.

See Also: For more information on how to use custom Work Managers to customize thread management, and when to use custom work managers, see the following:

- "Tune Pool Sizes" in *Tuning Performance of Oracle WebLogic Server*
- "Thread Management" in *Tuning Performance of Oracle WebLogic Server*
- "MDB Thread Management" in *Tuning Performance of Oracle WebLogic Server*
- "Using Work Managers to Optimize Scheduled Work" in *Administering Server Environments for Oracle WebLogic Server*
- "Avoiding and Managing Overload" in *Administering Server Environments for Oracle WebLogic Server*

You can use Oracle WebLogic Administration Console to view general information about the status of the thread pool (such as active thread count, total thread count, and queue length.) You can also use the Console to view each application's scoped work manager metrics from the Workload tab on the Monitoring page. The metrics provided include the number of pending requests and number of completed requests.

For more information, see "Servers: Monitoring: Threads" and "Deployments: Monitoring: Workload" in the *Oracle WebLogic Server Administration Console Online Help*.

The work manager and thread pool metrics can also be viewed from the Oracle Fusion Middleware Control.

2.10 Setting Logging Levels

The amount of information that applications log depends on how the environment is configured and how the application code is instrumented. To maximize performance it is recommended that the logging level is not set higher than the default `INFO` level logging. If the logging setting does not match the default level, reset the logging level to the default for best performance.

Once the application and server logging levels are set appropriately, ensure that the debugging properties or other application level debugging flags are also set to appropriate levels or disabled. To avoid performance impacts, do not set log levels to levels that produce more diagnostic messages, including the `FINE` or `TRACE` levels.

Each component may have specific recommendations for logging levels. See the component chapters in this book for more information.

3 Performance Planning

This chapter discusses performance and tuning concepts for Oracle Fusion Middleware. This chapter contains the following sections:

- Section 3.1, "About Oracle Fusion Middleware Performance Planning"
- Section 3.2, "Performance Planning Methodology"

3.1 About Oracle Fusion Middleware Performance Planning

To maximize Oracle Fusion Middleware performance, you must monitor, analyze, and tune all the components that are used by your applications. This guide describes the tools that you can use to monitor performance and the techniques for optimizing the performance of Oracle Fusion Middleware components.

Performance tuning usually involves a series of trade-offs. After you have determined what is causing the bottlenecks, you may have to modify performance in some other areas to achieve the expected results. However, if you have a clearly defined plan for achieving your performance objectives, the decision on what to trade for higher performance is easier because you have identified the most important areas.

3.2 Performance Planning Methodology

The Fusion Middleware components are built for performance and scalability. To maximize the performance capabilities of your applications, you must build performance and scalability into your design. The performance plan should address the current performance requirements, the existing issues (such as bottlenecks or insufficient hardware resources) and any anticipated variances in load, users or processes. The performance plan should also address how the components scale during peak usage without impacting performance.

The following sections of this chapter discuss the steps you should take to help create a plan to tune your application environment and optimize performance:

- Step 1: Define Your Performance Objectives
- Step 2: Design Applications for Performance and Scalability
- Step 3: Monitor and Measure Your Performance Metrics

3.2.1 Define Your Performance Objectives

Before you can begin performance tuning your applications, you must first identify the performance objectives you hope to achieve. To determine your performance objectives, you must understand the applications deployed and the environmental constraints placed on the system.

To understand what your performance objectives are, you must complete the following steps:

- Define Operational Requirements
- Identify Performance Goals
- Understand User Expectations
- Conduct Performance Evaluations

Performance objectives are limited by constraints, such as:

- The configuration of hardware and software such as CPU type, disk size, disk speed, and sufficient memory.

  There is no single formula for determining your hardware requirements. The process of determining what type of hardware and software configuration is required to meet application needs adequately is called *capacity planning*.

  Capacity planning requires assessment of your system performance goals and an understanding of your application. Capacity planning for server hardware should focus on maximum performance requirements.
- The configuration of high availability architecture to address peak usage and response times. For more information on implementing high availability features in Oracle Fusion Middleware applications, see the *High Availability Guide*.
- The ability to interoperate between domains, use legacy systems, support legacy data.
- Development, implementation, and maintenance costs.

Understanding these constraints - and their impacts - ensure that you set realistic performance objectives for your application environment, such as response times, throughput, and load on specific hardware.

3.2.1.1 Define Operational Requirements

Before you begin to deploy and tune your application on Oracle Fusion Middleware, it is important to clearly define the operational environment. The operational environment is determined by high-level constraints and requirements such as:

- Application Architecture
- Security Requirements
- Hardware Resources

3.2.1.2 Identify Performance Goals

Whether you are designing a new system or maintaining an existing system, you should set specific performance goals so that you know how and what to optimize. To determine your performance objectives, you must understand the application deployed and the environmental constraints placed on the system.

Gather information about the levels of activity that components of the application are expected to meet, such as:

- Anticipated number of users
- Number and size of requests

- Amount of data and its consistency
- Target CPU utilization

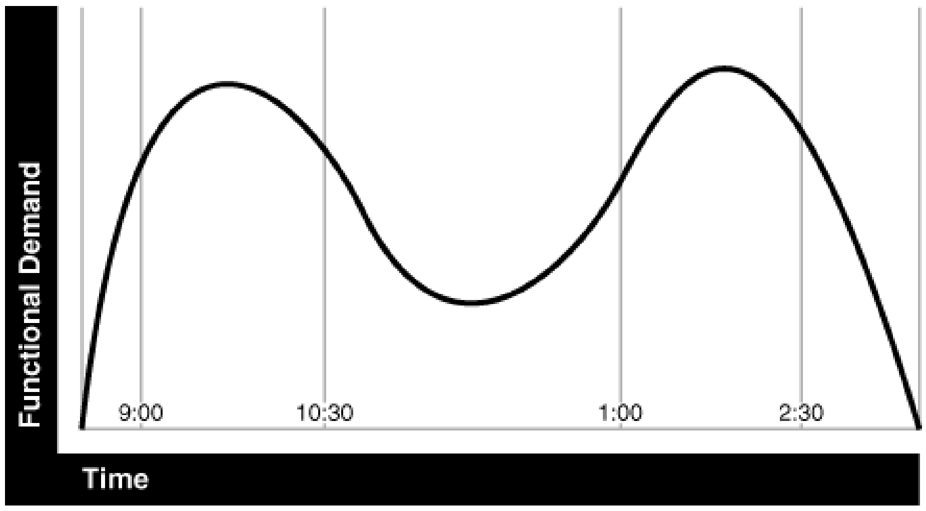
3.2.1.3 Understand User Expectations

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.

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 how variances in the load can affect the response time. 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 3–1. 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 can 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 3–1 Adjusting Capacity and Functional Demand



3.2.1.4 Conduct Performance Evaluations

With clearly defined performance goals and performance expectations, 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 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. For more information on evaluating performance, see Chapter 4, "Monitoring Oracle Fusion Middleware".

3.2.2 Design Applications for Performance and Scalability

The key to good performance is good design. The design phase of the application development cycle should be an on-going process. Cycling through the planning, monitoring and tuning phases of the application development cycle is critical to achieving optimal performance across Fusion Middleware deployments. Using an iterative design methodology enables you to accommodate changes in your work loads without impacting your performance objectives.

3.2.3 Monitor and Measure Your Performance Metrics

Oracle Fusion Middleware provides a variety of technologies and tools that can be used to monitor Server and Application performance. Monitoring enables you to evaluate Server activity, watch trends, diagnose system bottlenecks, debug applications with performance problems and gather data that can assist you in tuning the system. For more information, see Chapter 4, "Monitoring Oracle Fusion Middleware.".

Performance tuning is specific to the applications and resources that you have deployed on your system. Some common tuning areas are included in Chapter 2, "Top Performance Areas."

See Also: *Oracle Database Performance Tuning Guide*

Tuning Performance of Oracle WebLogic Server

Administering Oracle Fusion Middleware

4

Monitoring Oracle Fusion Middleware

Oracle Fusion Middleware provides a variety of technologies and tools that can be used to monitor Server and Application performance. Monitoring is an important step in performance tuning and enables you to evaluate server activity, watch trends, diagnose system bottlenecks, debug applications with performance problems and gather data that can assist you in tuning the system.

This chapter contains the following sections:

Note: Additional monitoring information is included for most products in the product-specific chapters of this guide.

4.1 About Oracle Fusion Middleware Management Tools

After you install and configure Oracle Fusion Middleware, you can use the graphical user interfaces or command-line tools to manage your environment.

Each tool is described in "Overview of Oracle Fusion Middleware Administration Tools" in *Administering Oracle Fusion Middleware*.

Note: The Oracle Process Manager and Notification Server (OPMN) is no longer used in Oracle Fusion Middleware. Instead, system components are managed by the WebLogic Management Framework, which includes WLST, Node Manager and pack and unpack. See "What Is the WebLogic Management Framework" in *Understanding Oracle Fusion Middleware*.

4.1.1 Measuring Your Performance Metrics

Metrics are the criteria you use to measure your scenarios against your performance objectives. You can use performance metrics to help locate bottlenecks, identify resource availability issues, or help tune your components to improve throughput and response times. After you have determined your performance criteria, take measurements of the metrics used to quantify your performance objectives.

For example, you might use response time, throughput, and resource utilization as your metrics. The performance objective for each metric is the value that is acceptable. You match the actual value of the metrics to your objectives to verify that you are meeting, exceeding, or failing to meet your performance objectives.

When you manage or monitor an Oracle Fusion Middleware component or application with Fusion Middleware Control, you may see performance metrics that provide insight into the current performance of the component or application. In many cases, these metrics are shown in interactive charts; other times they are presented in tabular format. The best way to use and correlate the performance metrics is from the Performance Summary page for the component or application you are monitoring.

The next sections of this chapter provide an overview of the Oracle Fusion Middleware technologies and tools that can be used to monitor Server and Application performance.

If you are new to Oracle Fusion Middleware or if you need additional information about monitoring your environment using the Performance Summary pages, see "Viewing the Performance of Oracle Fusion Middleware" in *Administering Oracle Fusion Middleware*. In addition, the Fusion Middleware Control online help provides definitions and other information about specific performance metrics that are available on its management and monitoring pages.

4.2 Oracle Enterprise Manager Fusion Middleware Control

Fusion Middleware Control is a Web browser-based, graphical user interface that you can use to monitor and administer your domain. It can manage an Oracle WebLogic Server domain with its Administration Server, one or more Managed Servers, clusters, the Oracle Fusion Middleware components that are installed, configured, and running in the domain, and the applications you deploy.

For more information, see "Getting Started Using Oracle Enterprise Manager Fusion Middleware Control" in *Administering Oracle Fusion Middleware*.

4.3 Oracle WebLogic Server Administration Console

Oracle WebLogic Server Administration Console is a Web browser-based, graphical user interface that you use to manage an Oracle WebLogic Server domain. It is accessible from any supported Web browser with network access to the Administration Server.

For more information on using the WebLogic Server console, see "Getting Started Using Oracle WebLogic Server Administration Console" in *Administering Oracle Fusion Middleware*.

Additional WebLogic Server Console Resources:

For details on the content contained in each summary table, see "Monitor Servers" in WebLogic Administration Console Online Help.

For detailed information on using the WebLogic Server to monitor your domain, see the *Tuning Performance of Oracle WebLogic Server*.

4.4 WebLogic Diagnostics Framework (WLDF)

The WebLogic Diagnostic Framework (WLDF) is a monitoring and diagnostic framework that can collect diagnostic data that servers and applications generate. The WLDF can be configured to collect the data and store it in various sources, including log records, data events, and harvested metrics.

For more information, see "Understanding the Diagnostic Framework" in *Administering Oracle Fusion Middleware*.

Note: For more information on the WebLogic Diagnostics Framework and how it can be leveraged for monitoring Oracle Fusion Middleware components, see *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

4.5 WebLogic Scripting Tool (WLST)

The Oracle WebLogic Scripting Tool (WLST) is a command-line scripting environment that you can use to create, manage, and monitor Oracle WebLogic Server domains. It is based on the Java scripting interpreter, Jython. In addition to supporting standard Jython features such as local variables, conditional variables, and flow-control statements, WLST provides a set of scripting functions (commands) that are specific to WebLogic Server. You can extend the WebLogic scripting language to suit your needs by following the Jython language syntax.

For more information, see "Getting Started Using the Oracle WebLogic Scripting Tool (WLST)" in *Administering Oracle Fusion Middleware*.

4.6 DMS Spy Servlet

The DMS Spy servlet provides access to DMS metric data from a web browser. Data that is created and updated by DMS-enabled applications and components is accessible through the DMS Spy Servlet.

4.6.1 Viewing Performance Metrics Using the Spy Servlet

The DMS Spy Servlet is part of the DMS web application. The DMS web application's web archive file is dms.war, and can be found in the same directory as `dms.jar`: `/modules/oracle.dms_12.1.2/dms.war`.

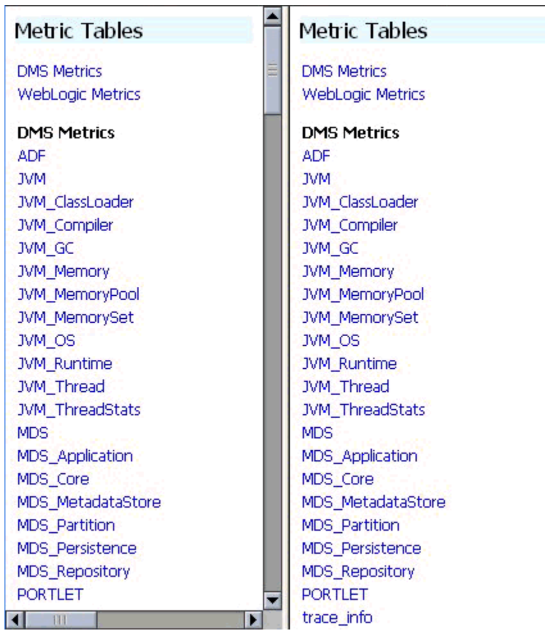
The DMS web application is deployed by default as part of a JRF-enabled server instance. The URL is: `http://host:port/dms/Spy`.

Only users who have Administrator role access can view this URL as access is controlled by standard Java EE elements in `web.xml`.

4.6.2 Using the DMS Spy Servlet

Figure 4–1 shows the initial page of the Spy servlet: both sides show the same list of metric tables.

Figure 4–1 Spy Servlet Page - Metrics Tables


Metric Tables

DMS Metrics
WebLogic Metrics

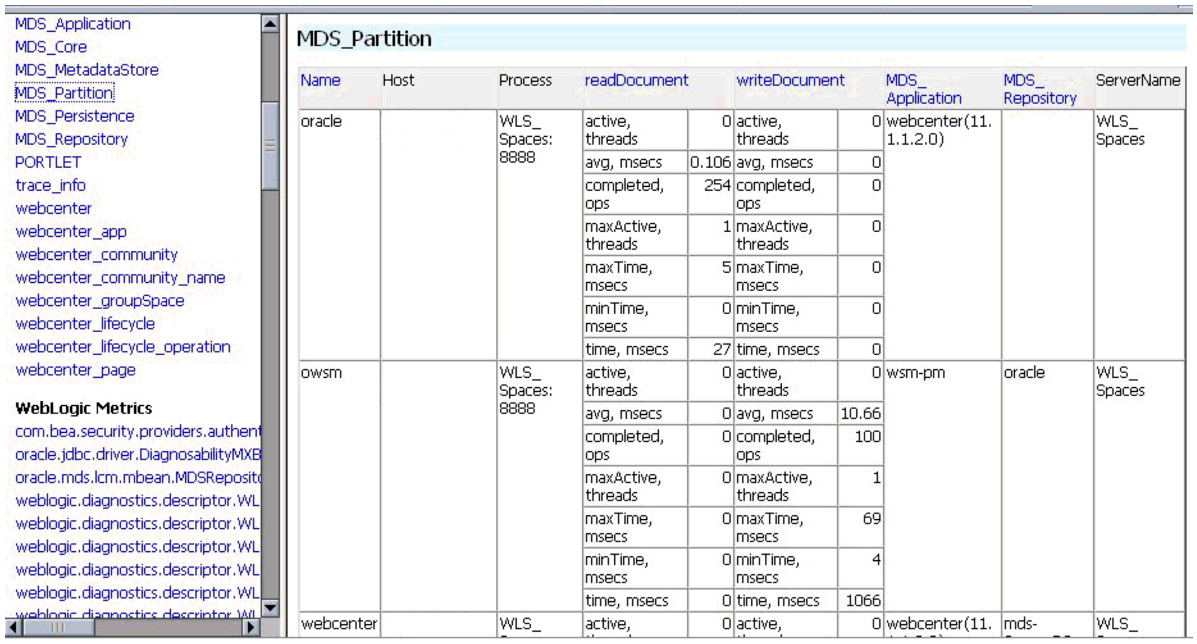
DMS Metrics
ADF
JVM
JVM_ClassLoader
JVM_Compiler
JVM_GC
JVM_Memory
JVM_MemoryPool
JVM_MemorySet
JVM_OS
JVM_Runtime
JVM_Thread
JVM_ThreadStats
MDS
MDS_Application
MDS_Core
MDS_MetadataStore
MDS_Partition
MDS_Persistence
MDS_Repository
PORTLET
trace_info

Note that the Spy servlet can display metric tables for WebLogic Server and also for non-Java EE components that are deployed.

For metric tables to appear in the Spy servlet, the component that creates and updates that table must be installed and running. Metric tables for components that are not running are not displayed. Metric tables with ":" in their name (for example, weblogic_j2eeserver:app_overview) are aggregated metric tables generated by metric rules.

To view the contents of a metric table, click the table name. For example, Figure 4–2 shows the MDS_Partition table.

Figure 4–2 MDS Partition Table



MDS_Application
MDS_Core
MDS_MetadataStore
MDS_Partition
MDS_Persistence
MDS_Repository
PORTLET
trace_info
webcenter
webcenter_app
webcenter_community
webcenter_community_name
webcenter_groupSpace
webcenter_lifecycle
webcenter_lifecycle_operation
webcenter_page

WebLogic Metrics
com.bea.security.providers.authent
oracle.jdbc.driver.DiagnosabilityMXB
oracle.mds.lcm.mbean.MDSReposit
weblogic.diagnostics.descriptor.WL
weblogic.diagnostics.descriptor.WL
weblogic.diagnostics.descriptor.WL
weblogic.diagnostics.descriptor.WL
weblogic.diagnostics.descriptor.WL

MDS_Partition

Name	Host	Process	readDocument		writeDocument		MDS_ Application	MDS_ Repository	ServerName
oracle		WLS_ Spaces: 8888	active, threads	0	active, threads	0	webcenter(11. 1.1.2.0)		WLS_ Spaces
			avg, msecs	0.106	avg, msecs	0			
			completed, ops	254	completed, ops	0			
			maxActive, threads	1	maxActive, threads	0			
			maxTime, msecs	5	maxTime, msecs	0			
			minTime, msecs	0	minTime, msecs	0			
			time, msecs	27	time, msecs	0			
owsm		WLS_ Spaces: 8888	active, threads	0	active, threads	0	wsm-pm	oracle	WLS_ Spaces
			avg, msecs	0	avg, msecs	10.66			
			completed, ops	0	completed, ops	100			
			maxActive, threads	0	maxActive, threads	1			
			maxTime, msecs	0	maxTime, msecs	69			
			minTime, msecs	0	minTime, msecs	4			
			time, msecs	0	time, msecs	1066			
webcenter		WLS_	active,	0	active,	0	webcenter(11.	mds-	WLS_

To get a description of the fields in a metric table, click the Metric Definitions link below the table.

4.7 Native Operating System Performance Commands

Each operating system has native tools and utilities that can be useful for monitoring purposes. Native operating system commands enable you to gather and monitor for example CPU utilization, paging activity, swapping, and other system activity information.

For details on operating system commands, refer to the documentation provided by the operating system vendor.

4.8 Network Performance Monitoring Tools

Your operating system's network monitoring tools can be used to monitor utilization, verify that the network is not becoming a bottleneck, or detect packet loss or other network performance issues. For details on network performance monitoring, refer to your operating system documentation.

5 Using the Oracle Dynamic Monitoring Service

This chapter provides an overview and features available in the Oracle Dynamic Monitoring Service (DMS).

- Section 5.1, "About Dynamic Monitoring Service (DMS)"
- Section 5.2, "Understanding DMS Availability"
- Section 5.3, "Understanding DMS Architecture"
- Section 5.4, "Viewing DMS Metrics"
- Section 5.5, "Accessing DMS Metrics with WLDF"
- Section 5.6, "DMS Execution Context"
- Section 5.7, "DMS Tracing and Events"
- Section 5.8, "DMS Best Practices"

5.1 About Dynamic Monitoring Service (DMS)

The Oracle Dynamic Monitoring Service (DMS) enables Oracle Fusion Middleware components to provide administration tools, such as Oracle Enterprise Manager, with data regarding the component's performance, state and on-going behavior. Fusion Middleware components push data to DMS and in turn DMS publishes that data through a range of different components. DMS measures and reports metrics, trace events and system performance and provides a context correlation service for these components.

5.1.1 Understanding Common DMS Terms and Concepts

This section defines common DMS terms and concepts related to the following:

- DMS Sensors
- DMS Nouns
- DMS Tracing and Events

5.1.1.1 DMS Sensors

DMS **sensors** measure performance data and enable DMS to define and collect a set of metrics. Certain metrics are always included with a sensor and other metrics are optionally included with a sensor.

DMS has three different kinds of sensors:

- Section 5.1.1.1.1, "DMS PhaseEvent Sensors"
- Section 5.1.1.1.2, "DMS Event Sensors"
- Section 5.1.1.1.3, "DMS State Sensors"

5.1.1.1.1 DMS PhaseEvent Sensors A DMS **PhaseEvent sensor** measures the time spent in a specific section of code that has a beginning and an end. Use a PhaseEvent sensor to track time in a method or in a block of code.

DMS can calculate optional metrics associated with a PhaseEvent, including the average, maximum, and minimum time that is spent in the PhaseEvent sensor.

Table 5–1 lists the metrics available with PhaseEvent sensors.

Table 5–1 DMS PhaseEvent Sensor Metrics

Metric	Description
`*sensor_name*.time`	Specifies the total time spent in the phase *sensor_name*. Default metric: `time` is a default PhaseEvent sensor metric.
`*sensor_name*.completed`	Specifies the number of times the phase *sensor_name* has completed since the process was started. Optional metric
`*sensor_name*.minTime`	Specifies the minimum time spent in the phase *sensor_name*, for all the times the *sensor_name* phase completed. Optional metric
`*sensor_name*.maxTime`	Specifies the maximum time spent in the phase *sensor_name*, for all the times the *sensor_name* phase completed. Optional metric
`*sensor_name*.avg`	Specifies the average time spent in the phase *sensor_name*, computed as the (total time)/(number of times the phase completed). Optional metric
`*sensor_name*.active`	Specifies the number of threads in the phase *sensor_name*, at the time the DMS statistics are gathered (the value may change over time). Optional metric
`*sensor_name*.maxActive`	Specifies the maximum number of concurrent threads in the phase *sensor_name*, since the process started. Optional metric

5.1.1.1.2 DMS Event Sensors A DMS **event sensor** counts system events. Use a DMS event sensor to track system events that have a short duration, or where the duration of the event is not of interest but the occurrence of the event is of interest.

Table 5–2 describes the metric that is associated with an event sensor.

Table 5–2 DMS Event Sensor Metrics

Metric	Description
`*sensor_name*.count`	Specifies the number of times the event has occurred since the process started, where *sensor_name* is the name of the Event sensor as specified in the DMS instrumentation API. Default: `count` is the default metric for an event sensor. No other metrics are available for an event sensor.

5.1.1.1.3 DMS State Sensors A DMS **state sensor** tracks the value of Java primitives or the content of a Java object. Supported types include integer, double, long, and object. Use a state sensor when you want to track system status information or when you need a metric that is not associated with an event. For example, use state sensors to track queue lengths, pool sizes, buffer sizes, or host names. You assign a precomputed value to a state sensor.

Table 5–3 describes the state sensor metrics. State sensors support a default metric `value`, as well as optional metrics. The optional `minValue` and `maxValue` metrics only apply for state sensors if the state sensor represents a numeric Java primitive (of type integer, double, or long).

Table 5–3 DMS State Sensor Metrics

Metric	Description
`sensor_name``.value`	Specifies the metric value for *`sensor_name`*, using the type assigned when *`sensor_name`* is created. Default: `value` is the default State metric.
`sensor_name``.count`	Specifies the number of times *`sensor_name`* is updated. Optional metric
`sensor_name``.minValue`	Specifies the minimum value for *`sensor_name`* since startup. Optional metric
`sensor_name``.maxValue`	Specifies the maximum value this *`sensor_name`* since startup. Optional metric

5.1.1.1.4 Sensor Naming Conventions The following list describes DMS sensor naming conventions:

- Sensor names should be descriptive, but not redundant. Sensor names should not contain any part of the noun name hierarchy, or type, as this is redundant.
- Sensor names should avoid containing the value for the individual metrics.
- Where multiple words are required to describe a sensor, the first word should start with a lowercase letter, and the following words should start with uppercase letters. Example: `computeSeries`
- In general, avoid using a "`/`" character in a sensor name. However, there are cases where it makes sense to use a name that contains "`/`". If a "`/`" is used in a noun or sensor name, then when you use the sensor in a string with DMS methods, you need to use an alternative delimiter, such as "," or "_", which does not appear anywhere in the path; this enables the "`/`" to be properly understood as part of the noun or sensor name rather than as a delimiter.

  For example, a child noun can have a name such as:

  ```
  examples/jsp/num/numguess.jsp
  ```

  and you can look this up using the string:

  ```
  ,default,WEBs,defaultWebApp,JSPs,example/jsp/num/numguess.jsp,service
  ```

  where the delimiter is the "," character.
- Event sensor and PhaseEvent sensor names should have the form *verbnoun*. Examples: `activateInstance` and `runMethod`. When a PhaseEvent monitors a function, method, or code block, it should be named to reflect the task performed as clearly as possible.

- The name of a state sensor should be a noun, possibly preceded by an adjective, which describes the semantics of the value which is tracked with this state sensor. Examples: `lastComputed`, `totalMemory`, `port`, `availableThreads`, `activeInstances`
- To avoid confusion, do not name sensors with strings such as ".time", ".value", or ".avg", which are names of sensor metrics, as shown in Table 5–1, Table 5–2, and Table 5–3.

5.1.1.2 DMS Nouns

DMS **nouns** organize performance data. Sensors, with their associated metrics, are organized in an hierarchy according to nouns. Nouns enable you to organize DMS metrics in a manner comparable to a directory structure in a file system. For example, nouns can represent classes, methods, objects, queues, connections, applications, databases, or other objects that you want to measure.

A **noun type** is the attribute that identifies the noun's type. Nouns that represent similar types of entities will typically have the same noun type and will usually record a common set of measurements for each of those entities.

5.1.1.2.1 General DMS Naming A **noun name** is a simple string, not including a delimiter. For example, `BasicBinomial` is a noun name. A noun full name consists of the noun name with the namespace and localpart. The noun name is preceded by the full name of its parent, and a delimiter. `/dmsDemo/BasicBinomial/"{http://mynamespace/}JAXWSHelloService"` is a noun full name.

A **sensor name** is a simple string, not including the "`.`" or the derivation. For example, `computeSeries`, `loops`, and `lastComputed` are sensor names.

A **sensor full name** consists of the sensor name, preceded by the name of its associated noun, and a delimiter. Examples: `/dmsDemo/BasicBinomial/computeSeries`, `/dmsDemo/BasicBinomial/loops`, `/dmsDemo/BasicBinomial/lastComputed`.

A **DMS metric name** consists of a sensor name plus the "`.`" character plus the metric. For example, `computeSeries.time`, `loops.count`, and `lastComputed.value` are valid DMS metric names.

Note: The suffixes .time, .count, and .value are immutable. Sensor and noun names, however, can be modified as needed.

5.1.1.2.2 General DMS Naming Conventions and Character Sets DMS names should be as compact as possible. When you define noun and sensor names, avoid special characters such as white space, slashes, periods, parenthesis, commas, and control characters.

Table 5–4 shows DMS replacement for special characters in names.

Table 5–4 Replacement for Special Characters in DMS Names

Character	DMS Replacement Character
Space character	Underscore character: _
Period character: .	Underscore character: _
Control character	Underscore character: _

Table 5–4 (Cont.) Replacement for Special Characters in DMS Names

Character	DMS Replacement Character
Less than character: `<`	Open parenthesis: `(`
Greater than character: `>`	Close parenthesis: `)`
Ampersand: `&`	Caret: `^`
Double quote: `"`	Backquote: `` ` ``
Single quote: `'`	Backquote: `` ` ``

Note: Oracle Fusion Middleware includes several built-in metrics. The Oracle Fusion Middleware built-in metrics do not always follow the DMS naming conventions.

5.1.1.2.3 Noun and Noun Type Naming Conventions The following conventions are used when naming noun and noun types:

- A noun name should be unique.
- A noun name should identify a specific entity of interest.
- Noun types should have names that clearly reflect the set of metrics being collected. For example, Servlet is the type for a noun under which the metrics that are specific to a given servlet fall.
- Noun type names should start with a capital letter to distinguish them from other DMS names. All nouns of a given type should contain the same set of sensors.
- The noun naming scheme uses a '/' as the root of the hierarchy, with each noun acting as a container under the root, or under its parent noun.

5.1.1.3 DMS Tracing and Events

Conceptually DMS generates a stream of events; each event is in response to one of the event-producing actions being performed on the DMS API by the components that integrate with DMS (such as a sensor being updated). That stream of events can be completely ignored or routed (and optionally filtered) to destinations that can respond in some way to events.

Table 5–5 provides a list of DMS tracing and event terminology.

Table 5–5 DMS Tracing and Event Terminology

DMS Term	Definition
Condition	A **condition** is the logic behind a condition filter. It determines which events may pass through a filter, based on the rules defined in the condition. Every condition filter has zero or one root condition, but conditions may include AND or OR arguments together to create compound conditions. The single root condition can describe a relatively complex rule. Two types of condition exist: ■ Noun Type Condition - operates on the name of the noun type associated with a sensor or noun event. ■ Context Condition - operates on the values currently set within the current Execution Context. For more information on using conditions, see Section 5.7, "DMS Tracing and Events".
Destination	A **destination** implements a mechanism for reacting to events that are passed to it. For example, a destination could log events to a file, another could send transformed copies of event to the Java Flight Recorder, yet another might render information gleaned from incoming events as data in an MBean.
Event Route	An **event route** connects a filter to a destination. Event routes may be enabled or disabled.
Filter	An event tracing **filter** selectively passes a subset of all possible DMS runtime events. Filters can be configured with rules that determine which events are passed and which are blocked. For example it is possible to define filters to: ■ Only pass sensor updates that are made when the execution context has a key-value pair of "role"-"admin" ■ Only pass sensor updates from nouns of type "JDBC_Statement" For more information on using filters, see Section 5.7, "DMS Tracing and Events".
Listener	A DMS **listener** is also known as the destination. See Section 5.7.2, "Configuring Destinations" for more information.

5.2 Understanding DMS Availability

DMS functionality is available on all certified Java EE servers. This includes both the runtime features and supporting commands. Also, several features of DMS will operate in JSE applications and standalone C applications.

For more information on which servers are certified, see the Oracle Fusion Middleware Certification Matrix.

5.3 Understanding DMS Architecture

DMS consists of the following features:

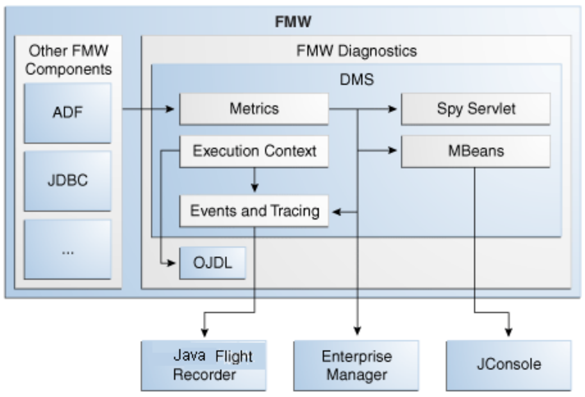
- **DMS Metrics** - The DMS metrics feature provides Java and C APIs that are used by Oracle Fusion Middleware components for instrumenting code with performance measurements and other useful state metrics.
- **Execution Context** - Execution Context supports the maintenance and propagation of a specific context structure throughout the Oracle stack. By exploiting the propagated context structure Oracle FMW components can record diagnostic information (such as log records) that can be correlated between different

components and products running on the same or different servers and hosts. For more information see Section 5.6, "DMS Execution Context".

- **Events and Tracing** - Event Tracing enables you to configure live tracing with no restarts. DMS metrics updated during the course of using Oracle Fusion Middleware products may be traced using the DMS Event Tracing feature. The system has been designed to facilitate not only tracing, but also to support other functionality that may be driven from DMS activity.

Figure 5–1 shows the components of DMS and how they interact with other Oracle Fusion Middleware components. Arrows show the direction in which information flows from one component to the next.

Figure 5–1 DMS Interactions with Oracle Fusion Middleware Components



5.4 Viewing DMS Metrics

Oracle Fusion Middleware components are instrumented with DMS metrics in order to collect information that developers, system administrators, and support analysts can use to analyze system performance or monitor system status. The Fusion Middleware Control online help provides information on each of the specific metrics. See "Viewing the Performance of Oracle Fusion Middleware" in *Administering Oracle Fusion Middleware* for information on accessing metric information.

The Oracle Fusion Middleware metrics come from various sources and locations. They include MBean attributes and DMS metrics. They also come from non-Java EE servers, such as Oracle HTTP servers.

The following sections describe how to use various tools to view the DMS metrics:

- Viewing Metrics Using the Spy Servlet
- Viewing Metrics with WLDF (WebLogic Diagnostic Framework)
- Viewing metrics with WLST (Oracle WebLogic Server)
- Viewing metrics with JConsole
- Viewing metrics with Oracle Enterprise Manager

5.4.1 Viewing Metrics Using the Spy Servlet

The Spy Servlet is part of the DMS Application that is deployed by default on JRF-extended installations. The Spy Servlet is launched from `http://<host>:<port>/dms/Spy`. The default port for WebLogic is 1521.

The DMS Application's web archive file is dms.war, and can be found in the same directory as `dms.jar`: `oracle_common/modules/oracle.dms_12.1.2/dms.war`.

For more information see Section 4.6, "DMS Spy Servlet".

Note: The Spy Servlet is secured using standard Java EE declarative security in the web-application's `web.xml` file, and will only grant access to the Spy Servlet to members of the Administrator's group.

5.4.2 Viewing Metrics with WLDF (WebLogic Diagnostic Framework)

You can use WebLogic Diagnostic Framework (WLDF) to harvest DMS metrics from DMS metric MBeans. You can also use WLDF to monitor changes to the attribute value of an MBean. For more information see "Configuring the Harvester for Metric Collection" in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

5.4.3 Viewing metrics with WLST (Oracle WebLogic Server)

DMS provides three commands to view metrics in WLST:

Use this command...	To do this...
`displayMetricTableNames()`	List the names of the available metric tables. If you have a large number of DMS metric tables, consider using the `outputfile` parameter with `displayMetricTableNames()`. This is useful when the output is expected to be large. When `displayMetricTableNames()` has the `outputfile` parameter, it returns null to the script instead of the whole output. This prevents the command from running out of memory. **NOTE**: In 12c, the command syntax for `displayMetricTableNames()` differs slightly for system components (such as OHS). After you connect WLST to Node Manager using `nmConnect()` command, you must specify both server name and server type explicitly. For example: `displayMetricTableNames(servertype="OHS", servers="ohs1")`

Use this command...	To do this...
`displayMetricTables()`	Show the content of the DMS metric tables. If you have a large number of DMS metric tables, consider using the `outputfile` parameter with `displayMetricTables()`. This is useful when the output is expected to be large. When `displayMetricTables()` has the `outputfile` parameter, it returns null to the script instead of the whole output. This prevents the command from running out of memory. **NOTE**: In 12c, the command syntax for `displayMetricTables()` differs slightly for system components (such as OHS). After you connect WLST to Node Manager using `nmConnect()` command, you must specify both server name and server type explicitly. For example: `displayMetricTables(servertype="OHS", servers="ohs1")`
`dumpMetrics()`	Display metrics in the internal format. Valid formats for the dumpMetrics command include raw, xml and pdml. If you have a large number of DMS metric tables, consider using the `outputfile` parameter with `dumpMetrics()`. This is useful when the output is expected to be large. When `dumpMetrics()` has the `outputfile` parameter, it returns null to the script instead of the whole output. This prevents the command from running out of memory. **NOTE**: In 12c, the command syntax for `dumpMetrics()` differs slightly for system components (such as OHS). After you connect WLST to Node Manager using `nmConnect()` command, you must specify both server name and server type explicitly. For example: `dumpMetrics()(servertype="OHS", servers="ohs1")`

As well as displaying textual output, theses commands also return a structured object or single value that you can use in a script to process.

For more information on using these commands, see the following:

- "Getting Started Using the Oracle WebLogic Scripting Tool (WLST)" in *Administering Oracle Fusion Middleware*
- "DMS Custom WLST Commands" in *WLST Command Reference for WebLogic Server*

5.4.4 Viewing metrics with JConsole

To provide a standards-based way to access metrics, DMS exposes them through MBeans. An MBean will be created and registered for each typed noun with the runtime MBean Server. The DMS sensors contained by the noun are exposed as the attributes of the MBean. Exposing the DMS metrics as MBeans allows administrators to use tools such as JConsole (the Java monitoring and management console), and other Java Management Extension (JMX) clients, to access the DMS metrics.

MBeans also allow for integration with other Oracle diagnostics software such as WLDF (WebLogic Diagnostics Framework), which is described in Section 5.5. The noun name and noun type are exposed as the name and type properties of the metric MBean object name. The MBean domain name is "oracle.dms". The object name also reflects the DMS noun hierarchy.

Note: You can use JConsole to view DMS generated MBeans on a Java EE server either locally or remotely. DMS generates an MBean for each Java DMS noun that has a valid noun type. It does not generate MBeans for the non-Java EE component's metrics and the DMS nouns that have no noun types. Each DMS metric contained under the noun is mapped to an attribute in the metric MBean.

5.4.5 Viewing metrics with Oracle Enterprise Manager

Oracle Fusion Middleware automatically and continuously measures data regarding the component's performance, state and on-going behavior. The metrics are automatically enabled; there is no need to set options or perform any extra configuration to collect them. For more information see Section 4.2, "Oracle Enterprise Manager Fusion Middleware Control".

5.5 Accessing DMS Metrics with WLDF

The WebLogic Diagnostics Framework (WLDF) provides a diagnostic feature that allows MBean attributes to be harvested and monitored for specific conditions. This provides a proactive way of monitoring activity in your environment and creating E-mail and JMX notifications when a condition is triggered.

The following steps describe how to configure WLDF to send an E-mail notification using the WebLogic Administration Console:

1. Select an existing or create a new Diagnostics Module from the Diagnostics screen.
2. Click on the **Watches and Notifications** tab.
3. Click **New**.
4. Enter a Watch Name and click **Next**.
5. Enter the text as the Watch Rule and click **Next**.

   ```
   (${ServerRuntime//[NOUNTYPE]oracle.dms:name=/starWars/alliance,type=NounType//f
   orceBalance_value} = 'BAD')
   ```

6. Select **Use a manual reset alarm** and click **Next**. The manual reset option means that once an E-mail is triggered, you must reset the watch using the WebLogic Administration Console.
7. Select the E-mail notification type and click **Finish**.

It is also possible to configure WLDF to collect the MBean data for offline storage and analysis. This is achieved by configuring a WLDF Diagnostic Module to collect specific MBean attributes, and can be done so using the WebLogic Administration Console.

For more information on using WLDF to harvest and monitor MBean data see *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

5.6 DMS Execution Context

The DMS execution context is the mechanism by which requests (such as HTTP or RMI requests) can be uniquely identified and thus tracked as they flow through the system. It also provides a means by which context information can be communicated between cooperating Fusion Middleware components involved in fulfilling requests.

5.6.1 DMS Execution Requests and Sub-Tasks

The DMS execution context has been developed with the understanding that a single request (or task) may form the root of a tree of sub-tasks that are coordinated to complete the request or root task. Consider the following examples of requests and their associated sub-tasks:

1. An HTTP request sent directly to Oracle WebLogic Server from a browser:
   - Root task only on Oracle WebLogic Server
2. An HTTP request sent through Oracle HTTP Server (acting as a reverse proxy) to Oracle WebLogic Server:
   - Root task on Oracle HTTP Server
   - Single sub-task on Oracle WebLogic Server
3. An HTTP request sent from Oracle HTTP Server (acting as a reverse proxy) to Oracle WebLogic Server that then requires invocation of two remote web services from Oracle WebLogic Server in order to fulfill the request:
   - Root task on Oracle HTTP Server
   - Single sub-task on Oracle WebLogic Server
   - Two sub-sub-tasks, one on each web service

A DMS execution context is composed of the following:

- A unique identifier, the ECID

  The Execution Context ID (ECID) is unique for each new root task and is shared across the tree of tasks associated with the root task.
- A relationship identifier, the RID

  The Relationship ID (RID) is an ordered set of numbers that describes the location of each task in the tree of tasks. The leading number is usually a zero. A leading number of 1 indicates that it has not been possible to track the location of the sub-task within the overall sub-task tree.
- A set of name-value pairs by which globally relevant data can be shared among Oracle Fusion Middleware components.

The following three scenarios illustrate how ECID and RID are used when an HTTP request is sent from Oracle HTTP Server (acting as a reverse proxy) to an Oracle WebLogic Server and the server requires invocation of two remote web services from Oracle WebLogic Server.

1. Root task on Oracle HTTP Server:
   – New ECID = B5C094FA...BE4AE8
   – Root RID = 0
2. Single sub-task on Oracle WebLogic Server:
   – Same ECID = B5C094FA...BE4AE8

- Sub-task RID = 0:1

3. Two Sub-tasks, one on each web service:
   - First web service invoked

     Same ECID = B5C094FA...BE4AE8

     Sub-task RID = 0:1:1
   - Second web service invoked

     Same ECID = B5C094FA...BE4AE8

     Sub-task RID = 0:1:2

5.6.2 DMS Execution Context Usage

The most immediate benefits of the DMS execution context are realized when attempting to correlate log messages between servers. The Oracle standard format for logging involves a field dedicated to the ECID. Once the ECID is known, when its read from an ERROR level log message for example, it is possible to locate all other log messages associated with that task by querying the log files for messages containing that ECID.

The following example shows a very specific case of using the command:

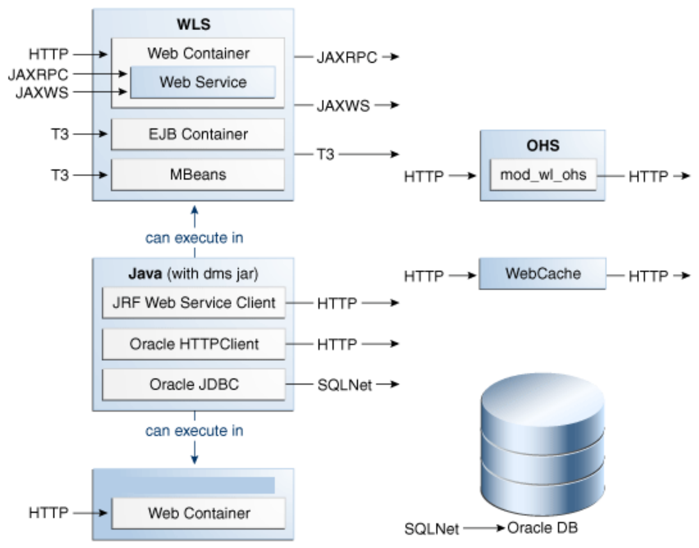
```
displayLogs(ecid="B5C094FA...BE4AE8");
```

In this example, any log files with messages that contain the ECID `B5C094FA...BE4AE8` will be displayed.

5.6.3 DMS Execution Context Communication

Figure 5–2 shows the components that cooperate in order to communicate the DMS execution context between each other. Arrows pointing to a component indicate the protocols that are inspected for incoming context information. Outgoing arrows show protocols to which context information is added. It is possible for a single component to send requests to itself, passing context information in that request.

Figure 5–2 DMS Execution Context Communication Protocols



5.7 DMS Tracing and Events

DMS can selectively trace the following:

- DMS sensor lifecycle events (create, update, delete of state sensors, event sensors and phase sensors)
- Context events (start, stop)
- HTTP events (start, stop)

The configuration that controls which of these types of events are traced, and how those events are processed, is recorded in the dms_config.xml file. The DMS trace configuration is split into three parts:

1. Filter Configuration

   Defines the rules that select the events that are of interest

2. Destination Configuration

   Defines how the events are used

3. eventRoute Configuration

   Defines which filters are wired to which destinations

A filter can be associated with one or more destinations thus granting the administrator the ability to define a filter rule once and have the resulting subset of all possible events processed on one or more different destinations.

The configuration can be modified using the DMS configuration MBean or WLST commands at runtime; this makes the DMS tracing feature invaluable for diagnosing issues within a specific time period or collecting specific data at a specific time for a specific set of criteria.

For more information, see "Configuring Selective Tracing Using WLST" in *Administering Oracle Fusion Middleware* .

The following types of filter rules are supported:

- Event Type Conditions

  Used to identify if an event was triggered from the START or STOP of a PHASE_SENSOR

- Context Type Conditions

  Used to identify if the event was generated from a unit of work whose context contains a value (for example, "USER")

- Noun Type Conditions

  Used to identify if the event was triggered from a sensor whose noun is of a specific type (for example, JDBC_CONNECTION

- Logical `AND` and `OR` combinations of the above conditions

5.7.1 Configuring the DMS Event System

Configuration is recorded in each server's dms_config.xml file. MBean updates can be made at runtime using command line interface (CLI) commands and through the Event Configuration Mbean. Configuration updates are applied to the running system in a thread safe, but non-atomic, manner.

The object name of the DMS Event configuration MBean is: `oracle.dms.event.config:name=DMSEventConfigMBean,type=JMXEventConfig`

To review the current state of your system's DMS event configuration, use the following command:

```
listDMSEventConfiguration([server=<server>])
```

The resulting output will look similar to this:

```
Event routes:
        FILTER      :  auto662515911
        DESTINATION :  destination1
        ENABLED     :  true
        FILTER      :  filter0
        DESTINATION :  q
        ENABLED     :  true
Filters with no event route:
  Fred

Destinations with no event route:
  des4
```

5.7.1.1 Adding and Editing Filters

Filters define the rules that select which events are considered for tracing.

The following example shows how to add a filter that selects all events related to JDBC operations:

```
addDMSEventFilter(id='myJDBCFilter', props={'condition': 'NOUNTYPE sw JDBC_'})
```

Or:

```
addDMSEventFilter(id='myJDBCFilter', props={'condition': 'NOUNTYPE startsWith
JDBC_'})
```

This filter assumes that all DMS sensor updates associated with JDBC operations are performed on nouns of types whose names begin "JDBC_".

If the rule must be modified, the filter may be updated as shown in the following example:

```
updateDMSEventFilter(id="myJDBCFilter", props={'condition': 'NOUNTYPE startsWith
JDBC_ OR NOUNTYPE startsWith MDS_'});
```

As of Oracle Fusion Middleware 11.1.1.6.0, the following shortened convenience operators have been added. Operators can be specified using either the shortened or longer name.

Note that operators with an underscore have been deprecated in favor of the ODL format, which is to use mixed case. For example, `not_equals` becomes `notEquals` or `ne`. The old format will still work, but is discouraged.

Noun Type Operators	
equals, eq	notEquals, ne
contains	in
startsWith, sw	

Context Operators	
equals, eq	notequals, ne
isnull	isnotnull
startswith, sw	contains
lt	gt

Example:

```
addDMSEventFilter(id='mdsbruce', name='MyFilter', props={'condition':
'NOUNTYPE eq MDS_Connections AND CONTEXT user ne bruce'})

addDMSEventFilter(id='mdsbruce', name='MyFilter', props={'condition':
'NOUNTYPE equals MDS_Connections AND CONTEXT user notequals bruce'})
```

For more information about the syntax used to describe a filter's rule (the condition property), refer to the WebLogic Scripting Tool Command Reference or the command help.

5.7.1.2 Adding and Editing Destinations

Destinations encapsulate logic for responding to events. For example, a basic destination will log the event, a different destination may transform an event and pass it to another system for further processing.

The following example shows how to add a destination that will log events:

```
addDMSEventDestination(id="myLoggerDestination",
class="oracle.dms.trace2.runtime.LoggerDestination",
props={"loggerName":"myLogger"});
```

Note that merely adding the destination is not sufficient for events to be logged; to log the events, you must associate a filter with a destination using an eventRoute, and the eventRoute must be enabled (default).

The types of destination available, and their configuration options, are described in Section 5.7.2. The following example shows how to edit an existing destination:

```
updateDMSEventDestination(id="myLoggerDestination",
props={"loggerName":"myTraceLogger"});
```

5.7.1.3 Adding and Editing Event Routes

The following example shows how to join the filter and destination created above:

```
addDMSEventRoute(filterid='myJDBCFilter', destinationid='myLoggerDestination')
```

Note that you can invoke `addDMSEventRoute` without an explicit filterId. In these scenarios, all events are passed to the destination without filtering.

To remove a filter or destination, you must first remove the event routes associated with the filter or destination (even if the event route is disabled). For example, if you wanted to remove `myJDBCFilter`, you would first need to remove the eventRoute created in the previous example, and then remove the filter as shown in the following example:

```
removeDMSEventRoute(filterid='myJDBCFilter', destinationid='myLoggerDestination')
removeDMSEventFilter(id='myJDBCFilter')
```

5.7.1.4 Compound Operations

It is possible to create a filter and an eventRoute based on that filter using a single command (rather than using two separate commands as shown in Section 5.7.1.3). Note, however, that the destination to be used by the event route must already be defined:

```
enableDMSEventTrace (destinationid='myLoggerDestination', condition='NOUNTYPE
starts_with JDBC_')
```

In the example above, `enableDMSEventTrace` automatically creates a filter with the specified condition, and also creates and enables an event route using the new filter and the nominated destination. The output is shown in the following example:

```
Filter "auto605449842" using Destination "myLoggerDestination" added, and
event-route enabled for server "AdminServer"
```

5.7.2 Configuring Destinations

DMS offers the following types of destinations:

- LoggerDestination
- MBean Creator Destination
- HTTP Request Tracker Destination
- Java Flight Recorder Destination

5.7.2.1 LoggerDestination

Description	The LoggerDestination writes each event to the associated logger.
Implementing Class	oracle.dms.trace2.runtime.LoggerDestination
Properties	
`loggerName`	The name of the ODL logger to which events will be written.

Instances of logger destinations write events to the named logger at a log level of FINER.

The `loggerName` property specifies the name of a logger, but the logger does not necessarily have to be described in logging.xml, though it can be. If the logger name refers to a logger that is explicitly named in logging.xml, then the logger is referred to as a static logger (see Section 5.7.2.1.1). If the logger name refers to a logger that is not explicitly named in logging.xml, then the logger is referred to as a dynamic logger (see Section 5.7.2.1.2).

Use in the default configuration: the default configuration defines a logger destination, with an identification of LoggerDestination. This particular instance does not form part of any eventRoute and therefore is not active. It is provided for convenience, and uses a dynamic logger.

5.7.2.1.1 Static Loggers and Handlers Loggers are the objects to which log records are presented. Log handlers are the objects through which log records are written to log files.

For complete control over the log file to which DMS trace data is written, define the logger named in the logger destination in logging.xml. Doing this allows you to explicitly define the name of the log file, the maximum size, format, file rotation and policies.

Oracle recommends using commands (like the example below) to update the configuration.

```
setLogLevel(logger="myTraceLogger", level="FINER", addLogger=1);

configureLogHandler(name="my-trace-handler", addToLogger=["myTraceLogger"],
path="/tmp/myTraceLogFiles/trace", maxFileSize="10m", maxLogSize="50m",
handlerType="oracle.core.ojdl.logging.ODLHandlerFactory", addHandler=1,
useParentHandlers=0);

configureLogHandler(name="my-trace-handler",
propertyName="useSourceClassandMethod", propertyValue="false", addProperty=1);
```

For more information on logging configuration, see "Managing Log Files and Diagnostic Data" in the *Administering Oracle Fusion Middleware*.]

The use of the optional property `useSourceClassandMethod` set to `FALSE` prevents the 'SRC_CLASS' and "SRC_METHOD' from appearing in every message and will marginally improve performance by reducing file output times.

For static loggers, consider setting the `useParentHandlers` parameter to `FALSE`, otherwise duplicate event messages will be logged to [server]-diagnostics.log, and shown in a log query.

See Section 5.7.3, "Understanding the Format of DMS Events in Log Messages" for more information about interpreting logger output.

5.7.2.1.2 Dynamic Loggers and Handlers If the named logger has no associated handler defined in logging.xml, then the logger destination will dynamically create a handler object that will write to a file in the server's default log output directory. (Instances of logger destinations write events to the named logger at a log level of FINER.) The file name will be the logger's name followed by "-event.log". For instance, in the example in Section 5.7.2.1.1, DMS events would be written to "myTraceLogger-event.log".

5.7.2.1.3 Default Locations of the logging.xml File The logging.xml file can typically be found in one of the following platform locations:

Platform	Server	Location
Oracle WebLogic Server	AdminServer	ORACLE_HOME/WLS_Home/user_projects/domains/base_domain/config/fmwconfig/servers/AdminServer/logging.xml

5.7.2.1.4 Using a CLI Command to Query the Trace Log File If the logger destination's logger and handler are defined in logging.xml then you can take advantage of the `displayLogs()` command to conveniently access logged trace data without having to manually locate or search for it.

Examples:

- To display all the log messages for the myTraceLogger:

  ```
  displayLogs(query='MODULE equals myTraceLogger')
  ```

- To display only the log messages for myTraceLogger which have an ECID of '0000HpmSpLWEkJQ6ub3FEH194kwB000004':

```
displayLogs(query='MODULE equals myTraceLogger and ECID equals
0000HpmSpLWEkJQ6ub3FEH194kwB000004')
```

- To display only the log messages for myTraceLogger which have an ECID of '0000HpmSpLWEkJQ6ub3FEH194kwB000004' in the last 10 minutes:

```
displayLogs(query='MODULE equals myTraceLogger and ECID equals
0000HpmSpLWEkJQ6ub3FEH194kwB000004', last=10)
```

- To display all the log messages from a dynamic logger the log's file name must be included:

```
displayLogs(disconnected=1, log=DOMAIN_
ROOT+"/servers/AdminServer/logs/myTraceLogger-event.log")
```

5.7.2.2 MBean Creator Destination

Description	The MBean creator destination make nouns accessible as MBeans, exposing their metrics as attributes, for access via WLDF, JConsole, etc.
Implementing Class	oracle.dms.jmx.MetricMBeanFactory

Use in the default configuration: An instance of the MBean Creator destination is configured and active by default, and will create MBeans for all nouns created in the server.

By associating an instance of this destination type with a filter based on a noun-type rule, it is possible to expose (as MBeans) only those noun types that are of interest to the administrator.

Although it is possible to modify the configuration associated with an MBean creator destination at runtime, it must be understood that the reinitialization process for this type of destination may impact performance. Frequent runtime reconfiguration is therefore discouraged.

Note that WebLogic Diagnostic Framework (WLDF) can be used to harvest DMS metrics exposed by the MBean creator destination. For more information about WLDF, see *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

5.7.2.2.1 Metric MBean Object Name The noun name and noun type are exposed as the name and type properties of the metric MBean object name. The MBean domain name is "oracle.dms". The object name also reflects the DMS noun hierarchy.

For example if the noun's full path name is:

```
 /oracle/dfw/ofm/base_domain/AdminServer
```

and the noun type is DFW_Incident, the object name of the MBean representing the noun is

```
oracle.dms:Location=AdminServer,name=/oracle/dfw/ofm/base_
domain/AdminServer,type=DFW_Incident.
```

5.7.2.3 HTTP Request Tracker Destination

Description	The HTTP Request Tracker destinations maintains a list of active HTTP requests, and makes the requests accessible to other Diagnostic Framework (DFW) components.
Implementing Class	oracle.dms.event.HTTPRequestTrackerDestination
Properties	
`excludeHeaderNames`	Comma separated list of header names to exclude from tracking

Use in the default configuration: An instance of the HTTP request tracker destination is enabled by default. In the case of a DFW incident being generated the active HTTP request list will be dumped automatically, allowing an administrator to correlate the failure with a specific request.

For each HTTP request the following information will be dumped:

- Uniform Resource Identifier (URI)
- Start time of the request
- Execution Context ID (ECID)
- Query string
- HTTP Headers

When the HTTP request tracker is not enabled the HTTP Request Dump will output the following:

```
HTTP Requests are not being tracked. To enable HTTP request tracking enable the
DMS oracle.dms.event.HTTPRequestTrackerDestination in dms_config.xml
```

5.7.2.3.1 Executing the HTTP Request Tracker Dump The information being maintained by the HTTP request tracker can be accessed manually. In order to execute the dump that reports the HTTP request information the WLST `executeDump` command can be used, when connected to a server, as follows:

```
> executeDump(name="http.requests")
Active Requests:

StartTime: 2009-12-14 02:24:41.870
ECID: 0000IMChyqEC8xT6uBf9EH1B9X9^000009,0
URI: /myApp/Welcome.jsp
QueryString:
Headers:
   Host: myHost.myDomain.com:7001
   Connection: keep-alive
   User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/532.5
(KHTML, like Gecko) Chrome/4.0.249.30 Safari/532.5
   Accept:
application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*
/*;q=0.5
   Accept-Encoding: gzip,deflate
   Cookie: ORA_MOS_LOCALE=en%7CGB; s_nr...
   Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
   Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
```

5.7.2.4 Java Flight Recorder Destination

The Java Flight Recorder (JFR) records information regarding the runtime status and behavior of the Java JVM. JFR also exposes an API through which third party events can be reported.

By themselves DMS traces and JFR traces only show part of the picture of the actions being performed in the server. DMS integration with JFR enhances the diagnostic information available to administrators and developers as follows:

1. Application level events and JVM level events can be reported as a single sequence therefore avoiding the need to combine such events from separate log files based only on timestamp (which may not tick over fast enough to accurately order events created at or around the same time).
2. Recent DMS activity can be dumped, retroactively, from the JVM at will.
3. Recent DMS and JVM events can be dumped to disk in the event of a fatal error that causes the JVM to exit gracefully.
4. The DMS ECID can be used to correlate activity relating to the same request, or unit of work, across the span of a JFR recording.
5. The DMS ECID can be used to collect diagnostic information from all systems involved with an event, or series of events, recorded by JFR.

5.7.2.4.1 Dynamically Derived JFR Event Types – Names, Values and Descriptions A DMS noun type will be associated with a JFR InstantEvent event type:

- The name of the JFR event type for a noun type will be the noun type's name with the suffix "state".
- The path of the JFR event type for a noun type will be "dms/" followed by the producer-name, followed by the event type name.
- Event sensors will not contribute any values to the noun type's JFR event type.
- The values of the JFR event for a noun type are described in Table 5–6:

Table 5–6 Values of the JFR Event for a Noun Type

Value Name	Description	Relational	Notes
ECID	The Execution Context ID (ECID) associated with the action.	Yes	
RID	The RID associated with the action.	Yes	

Table 5–6 (Cont.) Values of the JFR Event for a Noun Type

Value Name	Description	Relational	Notes
<noun type> name	The full path of the noun.		This field will be populated with the full path of the noun. The field's name assumes that the noun_type meaningfully categorizes all objects being measured by the nouns of that type.
<state-sensor-name>	The value of the state sensor.	No	Each state sensor belonging to the noun will contribute one of these values to the instant event. There may be more that one value in each noun.
event name	The name of the event sensor that was updated, left null otherwise.	No	The event name field is required for being able to count the number of times a DMS event sensor has been updated in a recording (event sensors do not contribute values to an event type).

A DMS phase sensor will be associated with a JFR DurationEvent event type:

- The name of the JFR event type for a phase sensor belonging to a noun of a particular noun type will be the noun type's name following by the phase sensor's name.
- The path of the JFR event for a noun type will be "dms/" followed by the producer-name, followed by the event type name.
- The values of the duration event will be as above (except for the sensorName value). For example the "stop" of a phase event will result in a JFR duration event being reported to JFR that contains the state information of the phase event's parent noun.

Several DMS objects allow integrators to add descriptions. Descriptions from DMS objects will be used as follows:

- Noun type description will be used in creation of the JFR event type
- State and event sensor descriptions will not be applied – there is nowhere to apply them.
- Phase sensor descriptions will be applied to their JFR event type.

5.7.2.4.2 Examples of Dynamically Derived Producers and Events Table 5–7 provides examples for the rules described in Section 5.7.2.4.1:

Table 5–7 Examples of Dynamically Derived Producers and Events

DMS	Java Flight Recorder (JFR)

Noun type:
`JDBC_Connection`
 	
Noun path:
`/JDBC/Driver/CONNECTION_7`
 	
Sensors:
`CreateStatement (P)`
`CreateNewStatement (P)`
`DBWaitTime (P)`
`JDBC_Connection_Url (S)`
`JDBC_Connection_Username (S)`
 	
Where:
P: Phase Sensor
S : State Sensor
E : Event Sensor
	Producer Name: JDBC
	The Producer Name is based on the leading component of the noun path.
	Event Type 1
	Event Type Name: `JDBC_Connection State`
	<noun type> State
	Event Type Path: `dms/JDBC/JDBC_Connection_State`
	dms/*<leading component of noun path>*/*<noun type>*/*_State*
	Fields:
	■ ECID
	■ RID
	■ JDBC_Connection name
	Value will be the full path of the noun
	■ JDBC_Connection_Url
	Value will be that of the state sensor of this name at the time of the event
	■ JDBC_Connection_Username
	Value will be that of the state sensor of this name at the time of the event
	■ Event Name
	Value will be one of the following:
	■ The name of the DMS event sensor whose activation caused this JFR event instance
	■ Null if this JFR event instance was created for a state sensor update

Table 5–7 (Cont.) Examples of Dynamically Derived Producers and Events

DMS	Java Flight Recorder (JFR)
	Producer Name: JDBC **Event Type 2** Event Type Name: `JDBC_Connection CreateStatement` Event Type Path: `dms/JDBC/JDBC_Connection_CreateStatement` **Fields:** ▪ ECID ▪ RID ▪ JDBC_Connection name ▪ JDBC_Connection_Url ▪ JDBC_Connection_Username
	Producer Name: JDBC **Event Type 3** Event Type Name: `JDBC_Connection CreateNewStatement` Event Type Path: `dms/JDBC/JDBC_Connection_CreateNewStatement` **Fields:** ▪ ECID ▪ RID ▪ JDBC_Connection name ▪ JDBC_Connection_Url ▪ JDBC_Connection_Username
	Producer Name: JDBC **Event Type 4** Event Type Name: `JDBC_Connection DBWaitTime` Event Type Path: `dms/JDBC/JDBC_Connection_DBWaitTime` **Fields:** ▪ ECID ▪ RID ▪ JDBC_Connection name ▪ JDBC_Connection_Url ▪ JDBC_Connection_Username

5.7.3 Understanding the Format of DMS Events in Log Messages

Table 5–8 describes the fields that make up a DMS event. Field elements are separated by ":" (with a few exceptions). Sample events are provided to illustrate the position of the field within an actual event string.

Table 5–8 Event Formatting Descriptions

Applicable Events	Field Number	Name	Description
All	1	Version number	The version number of the event format For example: **v1:**1280737384058:HTTP_REQUEST:STOP:/MyWebApp/emp
All	2	Event time	The time at which the event occurred For example: v1:**1280737384058**:HTTP_REQUEST:STOP:/MyWebApp/emp
All	3	Source object type	The type of object on which an action was performed to produce the event including: ■ NOUN ■ EVENT_SENSOR ■ STATE_SENSOR ■ PHASE_SENSOR ■ EXECUTION_CONTEXT ■ HTTP_REQUEST For example: v1:1280737384058:**HTTP_ REQUEST**:STOP:/MyWebApp/emp
All	4	Action type	The type of action that resulted in the generation of this event. A given source object type may not necessarily produce events for every action type: ■ CREATE ■ UPDATE ■ DELETE ■ START ■ STOP ■ ABORT For example: v1:1280737384058:HTTP_ REQUEST:**STOP**:/MyWebApp/emp
Nouns	5	Noun type	The name of the noun type For example: v1:1281344803506:NOUN:CREATE:**JDBC_ Connection**:/JDBC/JDBC Data Source-0/CONNECTION_1
	6	Noun path	The full path identifying the noun to which the sensor belongs For example: v1:1281344803506:NOUN:CREATE:JDBC_ Connection:**/JDBC/JDBC Data Source-0/CONNECTION_1**

Table 5–8 (Cont.) Event Formatting Descriptions

Applicable Events	Field Number	Name	Description
All Sensor Types	5	Noun type	The name of the noun type to which this sensor belongs For example: v1:1280503318973:STATE_SENSOR:UPDATE:**JDBC_ Connection**:LogicalConnection:/JDBC/JDBC Data Source-0/CONNECTION_ 1:State.ANY:LogicalConnection@13bed086
	6	Sensor name	The name of the sensor For example: v1:1280737383069:PHASE_SENSOR:STOP:JDBC_ Connection:**DBWaitTime**:/JDBC/JDBC Data Source-0/CONNECTION_1:1280737382950:1280737383069
	7	Noun path	The full path identifying the noun to which the sensor belongs For example: v1:1280737383069:PHASE_SENSOR:STOP:JDBC_ Connection:DBWaitTime:**/JDBC/JDBC Data Source-0/CONNECTION_1**:1280737382950:1280737383069
Phase Sensor Types	8	Start token	The start token of the phase. For example: v1:1280737383069:PHASE_SENSOR:STOP:JDBC_ Connection:DBWaitTime:/JDBC/JDBC Data Source-0/CONNECTION_1:**1280737382950**:1280737383069
	9	Stop token	The end token of the phase. For example: v1:1280737383069:PHASE_SENSOR:STOP:JDBC_ Connection:DBWaitTime:/JDBC/JDBC Data Source-0/CONNECTION_1:1280737382950:**1280737383069**

Table 5–8 (Cont.) Event Formatting Descriptions

Applicable Events	Field Number	Name	Description
State Sensor Types	8	State value type	The type of value held by the state sensor including: ■ State.DOUBLE ■ State.INTEGER ■ State.LONG ■ State.OBJECT ■ State.ANY For example: v1:1280503318973:STATE_SENSOR:UPDATE:JDBC_ Connection:LogicalConnection:/JDBC/JDBC Data Source-0/CONNECTION_ 1:**State.ANY**:LogicalConnection@13bed086
	9	State value	The value of the state represented in string form. For example: v1:1280503318973:STATE_SENSOR:UPDATE:JDBC_ Connection:LogicalConnection:/JDBC/JDBC Data Source-0/CONNECTION_ 1:State.ANY:**LogicalConnection@13bed086**
HTTP Requests	5	URI	Uniform Resource Identifier (URI) identifies the resource upon which to apply the request. For example: v1:1280737382889:HTTP_ REQUEST:START:**/myWebApp/showEmployees** v1:1280737384058:HTTP_ REQUEST:STOP:**/myWebApp/showEmployees**
Execution Context	5	ECID,RID	The context identifier (composed of ECID and RID separated by a comma). For execution context events the complete substring starting at the first character after the fourth event field separator (":") records the ECID,RID identifiers - the context identifiers may contain ":" but these should not be interpreted as event field separators. For example: v1:1280737384058:EXECUTION_ CONTEXT:STOP:**bc4fd0668f79d507:367c127f:12a23f2013c:-8000-0000000000000f73,0**

5.7.4 Understanding DMS Event Actions

Table 5–9 shows the action types that can be performed on source object types.

Table 5–9 Actions Performed on Source Object Types

	Create	Update	Delete	Start	Stop	Abort
Noun	Yes	-	Yes	-	-	-
Event Sensor	Yes	Yes	Yes	-	-	-
Phase Sensor	Yes	-	Yes	Yes	Yes	Yes
State Sensor	Yes	Yes	Yes	-	-	-

Table 5–9 (Cont.) Actions Performed on Source Object Types

	Create	Update	Delete	Start	Stop	Abort
Execution Context	-	-	-	Yes	Yes	-
Http Request	-	-	-	Yes	Yes	-

5.8 DMS Best Practices

The use of DMS metrics can have an impact on application performance. When adding metrics, consider the following:

- Use a High Resolution Clock to increase DMS Precision

  By default DMS uses the system clock for measuring time intervals during a `PhaseEvent`. The default clock reports microsecond precision in C processes such as Apache and reports millisecond precision in Java processes. Optionally, DMS supports a high resolution clock to increase the precision of performance measurements and lets you select the values for reporting time intervals. You can use a high resolution clock when you need to time phase events more accurately than is possible using the default clock or when the system's default clock does not provide the resolution needed for your requirements.

  System clocks are not necessarily as accurate as their precision implies. For example, a system clock that reports time in milliseconds may not tick (change) once per millisecond. Instead, it may take up to 15ms to tick as shown in the following example:

Table 5–10 Default System Clock Time versus Actual Time (in milliseconds)

Actual Time	System Time
12:00:00.000	12:00:00.000
12:00:00.001	12:00:00.000
12:00:00.002	12:00:00.000
[...]	
12:00:00.014	12:00:00.000
12:00:00.015	12:00:00.015
12:00:00.016	12:00:00.015

  Table 5–10 shows a phase with a 12ms duration that runs from actual time 12:00:00.002 to 12:00:00.014 would be calculated in system time as having a duration of zero. Similarly, a phase with a 2ms duration running from 12:00:00.014 to 12:00:00.016 would be reported in system time as having a duration of 15ms.

  Note: These behaviors are more evident on some operating systems than others. Use caution when analyzing individual periods of time that are shorter than the tick period of the system clock. Configuring DMS to use a higher resolution clock will cause DMS to record phase sensor activations with higher resolution, but the accuracy will still be limited by the underlying system.

- Configure DMS Clocks for Reporting Time for Java

Selecting the high resolution clock changes clocks for all applications running on the server where the clock is changed. You set the DMS clock and the reporting values globally using the `oracle.dms.clock` and `oracle.dms.clock.units` properties, which control process startup options.

For example, to use the high resolution clock with the default values, set the following property on the Java command line:

```
-Doracle.dms.clock=highres
```

Caution: If you use the high resolution clock, the default values are different from the value that Fusion Middleware Control expects (msecs). If you need the Fusion Middleware Control displays to be correct when using the high resolution clock, then you need to set the units property as follows:

```
-Doracle.dms.clock.units=msecs
```

Table 5–11 shows supported values for the `oracle.dms.clock` property.

Table 5–12 shows supported values for the `oracle.dms.clock.units` property.

Table 5–11 oracle.dms.clock Property Values

Value	Description
DEFAULT	Specifies that DMS use the default clock. With the default clock, DMS uses the Java call `java.lang.System.currentTimeMillis` to obtain times for PhaseEvents. The default value for the units for the default clock is MSECS.
HIGHRES	The Java Highres clock uses `System.nanoTime()` (no JNI required).

Table 5–12 oracle.dms.clock.units Property Values

Value	Description
MSECS	Specifies that the time be converted to milliseconds and reported as "msecs". A millisecond is 10^{-3} seconds. **Note**: This is the default value for the default clock.
USECS	Specifies that the time be converted to microseconds and reported as "usecs". A microsecond is 10^{-6} seconds.
NSECS	Specifies that the time be converted to nanoseconds and reported as "nsecs". A nanosecond is 10^{-9} seconds. **Note**: This is the default value for the high resolution clock.

Note the following when using the high resolution DMS clock:

- When you set the `oracle.dms.clock` and the `oracle.dms.clock.units` properties, any combination of upper and lower case characters is valid for the value that you select (case is not significant). For example, any of the following values are valid to select the high resolution clock: highres, HIGHRES, HighRes.
- DMS checks the property values at startup. When the clock property is set with a value not listed in Table 5–11, DMS uses the default clock. If the `oracle.dms.clock` property is not set, DMS uses the default clock.

- When the clock units property is set to a value not listed in Table 5–12, DMS uses the default units for the specified clock.

Part II

Core Components

This part describes configuring core components to improve performance. It contains the following chapters:

Note: For information on performance tuning the Oracle WebLogic Server, see *Tuning Performance of Oracle WebLogic Server*.

6 Oracle HTTP Server Performance Tuning

This chapter discusses the techniques for optimizing Oracle HTTP Server performance. This chapter contains the following sections:

- Section 6.1, "About Oracle HTTP Server"
- Section 6.2, "Monitoring Oracle HTTP Server Performance"
- Section 6.3, "Basic Tuning Considerations"
- Section 6.4, "Advanced Tuning Considerations"

Note: The configuration examples and recommended settings described in this chapter are for illustrative purposes only. Consult your own use case scenarios to determine which configuration options can provide performance improvements.

6.1 About Oracle HTTP Server

Oracle HTTP Server (OHS) is the Web server component for Oracle Fusion Middleware. It provides a listener for Oracle WebLogic Server and the framework for hosting static pages, dynamic pages, and applications over the Web. Oracle HTTP Server is based on the Apache 2.2.x infrastructure, and includes modules developed specifically by Oracle. The features of single sign-on, clustered deployment, and high availability enhance the operation of the Oracle HTTP Server.

For more information see "Introduction to Oracle HTTP Server" *Administering Oracle HTTP Server*.

For more information on the Apache open-source software infrastructure, see the Apache Software Foundation web site at `http://www.apache.org/`.

6.2 Monitoring Oracle HTTP Server Performance

Oracle Fusion Middleware automatically and continuously measures run-time performance for Oracle HTTP Server. The performance metrics are automatically enabled; you do not need to set options or perform any extra configuration to collect them. If you encounter a problem, such as an application that is running slowly or is hanging, you can view particular metrics to find out more information about the problem.

Note: Fusion Middleware Control provides real-time data. For more information on using Fusion Middleware Control to view performance metrics for HTTP Server, see "Managing and Monitoring Server Processes" in *Administering Oracle HTTP Server*.

For monitoring, Oracle HTTP Server uses the Dynamic Monitoring Service (DMS), which collects metrics for every functional piece. You can review these metrics as needed to understand system behavior at a given point of time. This displays memory, CPU information and the minimum, maximum, and average times for the request processing at every layer in Oracle HTTP Server. The metrics also display details about load level, number of threads, number of active connections, and so on, which can help in tuning the system based on real usage.

For more information on using these DMS metrics, see Section 5.4.3, "Viewing metrics with WLST (Oracle WebLogic Server)".

6.3 Basic Tuning Considerations

The following tuning configurations may improve the performance of the Oracle HTTP Server. Always consult your own use case scenarios to determine if these settings are applicable to your deployment.

- Tuning Oracle HTTP Server Directives
- Reducing Httpd Process Availability with Persistent Connections
- Logging Options for Oracle HTTP Server

6.3.1 Tuning Oracle HTTP Server Directives

Oracle HTTP Server uses directives in `httpd.conf`. This configuration file specifies the maximum number of HTTP requests that can be processed simultaneously, logging details, and certain limits and time outs.

More information on configuring the Oracle HTTP Server, see "Management Tools for Oracle HTTP Server" in *Administering Oracle HTTP Server*.

Oracle HTTP Server supports three different Multi-Processing Modules (MPMs) by default. The MPMs supported are:

- Worker - This uses Multi-Process-Multi-Threads model and is the default MPM on all platforms other than Microsoft Windows platforms. Multi-thread support makes it more scalable by using fewer system resources and multi-process support makes it more stable.
- WinNT - This MPM is for Windows platforms only. It consists of a parent process and a child process. The parent process is the control process, and the child process creates threads to handle requests.
- Prefork - This is Apache 1.3.x style and uses processes instead of threads. This is considered the least efficient MPM.

The directives for each MPM type are defined in the `ORACLE_INSTANCE/config/OHSComponent/<ohsname>/httpd.conf file`. The default MPM type is Worker MPM. To use a different MPM (such as Prefork MPM), edit the `/ohs/bin/apachectl` file.

Note: The information in this chapter is based on the use of Worker and WinNT MPMs, which use threads. The directives listed below may not be applicable if you are using the prefork MPM. If you are using Oracle HTTP Server based on Apache 1.3.x or Apache 2.2 with prefork MPM, refer to the Oracle Application Server 10*g* Release 3 documentation at `http://www.oracle.com/technology/documentation/appserver10132.html`.

Table 6–1 Oracle HTTP Server Configuration Properties

Directive	Description
`ListenBackLog` This directive maps to the **Maximum Queue Length** field on the Performance Directives screen.	Specifies the maximum length of the queue of pending connections. Generally no tuning is needed. Note that some operating systems do not use exactly what is specified as the backlog, but use a number based on, but normally larger than, what is set. Default Value: 511
`MaxClients` This directive maps to the **Maximum Requests** field on the Performance Directives screen. Note that this parameter is not available in mod_ winnt (Microsoft Windows). Winnt uses a single process, multi-threaded model and is controlled by `ThreadLimit` directive.	Specifies a limit on the total number of servers running, that is, a limit on the number of clients who can simultaneously connect. If the number of client connections reaches this limit, then subsequent requests are queued in the TCP/IP system up to the limit specified with the `ListenBackLog` directive (after the queue of pending connections is full, new requests generate connection errors until a thread becomes available). You can configure the `MaxClients` directive in the httpd.conf file up to a maximum of 8000 (8K) (the default value is 150). If your system is not resource-saturated and you have a user population of more than 150 concurrent HTTP/Thread connections, you can improve your performance by increasing `MaxClients` to increase server concurrency. Increase `MaxClients` until your system becomes fully utilized (85% is a good threshold). Conversely, when system resources are saturated, increasing `MaxClients` does not improve performance. In this case, the `MaxClients` value could be reduced as a throttle on the number of concurrent requests on the server. If the server handles persistent connections, then it may require sufficient concurrent httpd or thread server processes to handle both active and idle connections. When you specify `MaxClients` to act as a throttle for system concurrency, you must consider that persistent idle httpd connections also consume httpd/thread processes. Specifically, the number of connections includes the currently active persistent and non-persistent connections and the idle persistent connections. A persistent, `KeepAlive`, http connection consumes an httpd child process, or thread, for the duration of the connection, even if no requests are currently being processed for the connection. If you have sufficient capacity, `KeepAlive` should be enabled; using persistent connections improves performance and prevents wasting CPU resources reestablishing HTTP connections. Normally, you should not change `KeepAlive` parameters. The maximum allowed value for `MaxClients` is 8192 (8K). Default Value: 150
`StartServers` This directive maps to the **Initial Child Server Processes** field on the Performance Directives screen.	Specifies the number of child server processes created on startup. If you expect a sudden load after restart, set this value based on the number child servers required. Note that the following parameters are inter-related and applicable only on UNIX platforms (worker_mpm): ■ `MaxClients` ■ `MaxSpareThreads` and `MinSpareThreads` ■ `ServerLimit` and `StartServers` On the Windows platform (mpm_winnt), as well as UNIX platforms, the following parameters are important to tune: ■ `ThreadLimit` ■ `ThreadsPerChild` Note that each child process has a set of child threads defined for them and that can actually handle the requests. Use `ThreadsPerChild` in connection with this directive. The values of `ThreadLimit`, `ServerLimit`, and `MaxClients` can indirectly affect this value. Read the notes for these directives and use them in conjunction with this directive. Default Value: 2

Table 6–1 (Cont.) Oracle HTTP Server Configuration Properties

Directive	Description
`ServerLimit` Note that this parameter is not available in mod_ winnt (Microsoft Windows). Winnt uses a single process, multi-threaded model	Specifies an upper limit on the number of server (child) processes that can exist or be created. This value overrides the `StartServers` value if that value is greater than the `ServerLimit` value. This is used to control the maximum number of server processes that can be created. Default Value: 16
`ThreadLimit`	Specifies the upper limit on the number of threads that can be created under a server (child) process. This value overrides the `ThreadsPerChild` value if that value is greater than the `ThreadLimit` value. This is used to control the maximum number of threads created per process to avoid conflicts/issues. Default Values: ■ Windows Multi-Processing Module (mpm_winnt): 1920 ■ All others: 64
`ThreadsPerChild` This directive maps to the **Threads Per Child Server Process** field on the Performance Directives screen.	Sets the number of threads created by each server (child) process at startup. Default Value: 64 when mpm_winnt is used and 25 when Worker MPM is used. The `ThreadsPerChild` directive works with other directives, as follows: At startup, Oracle HTTP Server creates a parent process, which creates several child (server) processes as defined by the `StartServers` directive. Each server process creates several threads (server/worker), as specified in `ThreadsPerChild`, and a listener thread which listens for requests and transfers the control to the worker/server threads. After startup, based on load conditions, the number of server processes and server threads (children of server processes) in the system are controlled by `MinSpareThreads` (minimum number of idle threads in the system) and `MaxSpareThreads` (maximum number of idle threads in the system). If the number of idle threads in the system is more than `MaxSpareThreads`, Oracle HTTP Server terminates the threads and processes if there are no child threads for a process. If the number of idle threads is fewer than `MinSpareThreads`, it creates new threads and processes if the `ThreadsPerChild` value has already been reached in the running processes. The following directives control the limit on the above directives. Note that the directives below should be defined before the directives above for them to take effect. ■ `ServerLimit` - Defines the upper limit on the number of servers that can be created. This affects `MaxClients` and `StartServers`. ■ `ThreadLimit` - Defines the upper limit on `ThreadsPerChild`. If `ThreadsPerChild` is greater than `ThreadLimit`, then it is automatically trimmed to the latter value. ■ `MaxClients` - Defines the upper limit on the number of server threads that can process requests simultaneously. This should be equal to the number of simultaneous connections that can be made. This value should be a multiple of `ThreadsPerChild`. If `MaxClients` is greater than `ServerLimit` multiplied by `ThreadsPerChild`, it is automatically be trimmed to the latter value.

Table 6–1 (Cont.) Oracle HTTP Server Configuration Properties

Directive	Description
`MaxRequestsPerChild` This directive maps to the **Max Requests Per Child Server Process** field on the Performance Directives screen.	Specifies the number of requests each child process is allowed to process before the child process dies. The child process ends to avoid problems after prolonged use when Apache (and any other libraries it uses) leak memory or other resources. On most systems, this is not needed, but some UNIX systems have notable leaks in the libraries. For these platforms, set `MaxRequestsPerChild` to 10000; a setting of 0 means unlimited requests. 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 "keep alive" requests, it would only count as 1 request toward this limit. Default Value: 0 **Note**: On Windows systems `MaxRequestsPerChild` should always be set to `0` (unlimited) since there is only one server process.

`MaxSpareThreads`
 	
`MinSpareThreads`
 	
These directives map to the **Maximum Idle Threads** and **Minimum Idle Threads** fields on the Performance Directives screen.
 	
Note that these parameters are not available in mod_ winnt (Windows platform).
	Controls the server-pool size. Rather than estimating how many server threads you need, Oracle HTTP Server dynamically adapts to the actual load. The server tries to maintain enough server threads to handle the current load, plus a few additional server threads to handle transient load increases such as multiple simultaneous requests from a single browser.
	
	The server does this by periodically checking how many server threads are waiting for a request. If there are fewer than `MinSpareThreads`, it creates a new spare. If there are more than `MaxSpareThreads`, some of the spares are removed.
	
	Default Values:
	
	MaxSpareThreads: 75
	
	MinSpareThreads: 25

`Timeout` This directive maps to the **Request Timeout** field on the Performance Directives screen.	The number of seconds before incoming receives and outgoing sends time out. Default Value: 300
`KeepAlive` This directive maps to the **Multiple Requests Per Connection** field on the Performance Directives screen.	Whether or not to allow persistent connections (more than one request per connection). Set to Off to deactivate. Default Value: On

Table 6–1 (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, consider increasing this value. Default Value: 100
`KeepAliveTimeout` This directive maps to the **Allow With Connection Timeout (seconds)** field, which is located under the **Multiple Requests Per Connection** field, on the Performance Directives screen.	Number of seconds to wait for the next request from the same client on the same connection. Default Value: 5 seconds
`limit` `ulimit`	Number of objects that a program uses to read or write to an open file or open network sockets. A lack of available file descriptors can impact operating system performance. Tuning the file descriptor limit can be accomplished by configuring the hard limit (`ulimit`) in a shell script which starts the OHS. Once the hard limit has been set the OHS will then adjust the soft limit (`limit)` to match. Note that configuring file descriptor limits is platform specific. Refer to your operating system documentation for more information.

6.3.2 Reducing Httpd Process Availability with Persistent Connections

If your browser supports persistent connections, you can support them on the server using the `KeepAlive` directives in the Oracle HTTP Server. Persistent Connections can improve performance by reducing the work load on the server. With Persistent Connections enabled, the server does not have to repeat the work to set up the connections with a client.

The default settings for the `KeepAlive` directives are:

```
KeepAlive on
MaxKeepAliveRequests 100
KeepAliveTimeOut 5
```

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 when setting these values. For example, if you have a large user population and the users make small infrequent requests, you may want to reduce the `keepAlive` directive default 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.

`KeepAlive` option should be used judiciously along with `MaxClients` directive. `KeepAlive` option would tie a worker thread to an established connection until it times out or the number of requests reaches the limit specified by `MaxKeepAliveRequests`. This means that the connections or users in the `ListenBacklog` queue would be starving for a worker until the worker is relinquished by the keep-alive user. The starvation for resources happens on the `KeepAlive` user load with user population consistently higher than that specified in the `MaxClients`.

Note: The `Maxclients` property is applicable only to UNIX platforms. On Windows, the same functionality is achieved through the `ThreadLimit` and `ThreadsPerChild` parameters.

Increasing `MaxClients` may impact performance in the following ways:

- A high number of `MaxClients` can overload the system resources and may lead to poor performance.
- For a high user population with fewer requests, consider increasing the `MaxClients` to support `KeepAlive` connections to avoid starvation. Note that this can impact overall performance if the user concurrency increases. System performance is impacted by increased concurrency and can possibly cause the system to fail.

`MaxClients` should always be set to a value where the system would be stable or performing optimally (~85% CPU).

Typically for high user population with less frequent requests, consider turning the `KeepAlive` option off or reduce it to a very low value to avoid starvation.

Disabling the `KeepAlive` connection may impact performance in the following ways:

- Connection establishment for every request has a cost.
- If the frequency of creating and closing connections is higher, then some system resources are used. The TCP connection has a `time_wait` interval before it can close the socket connection and open file descriptors for every connection. The default `time_wait` value is 60 seconds and each connection can take 60 seconds to close, even after it is relinquished by the server.

WARNING: To avoid potential performance issues, values for any parameters should be set only after considering the nature of the workload and the system capacity.

6.3.3 Logging Options for Oracle HTTP Server

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

6.3.3.1 Access Logging

Access logs are generally enabled to track who accessed what. The access_log file, available in the ORACLE_INSTANCE/diagnostics/logs/OHS/*ohsname* directory, contains an entry for each request that is processed. This file grows as time passes and can consume disk space. Depending on the nature of the workload, the access_log has little impact on performance. If you notice that performance is becoming an issue, the file can be disabled if some other proxy or load balancer is used and gives the same information.

6.3.3.2 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. Depending on the server load and the network connectivity to your DNS server, the performance impact of the DNS HostNameLookup may be high. When possible, consider logging only IP addresses. On UNIX systems, you can resolve IP addresses to host names off-line, with the `logresolve` utility found in the `/Apache/Apache/bin/` directory.

6.3.3.3 Error logging

The server notes unusual activity in an error log. The *ohsname*.log file, available in ORACLE_INSTANCE/diagnostics/logs/OHS/*ohsname* directory, contains errors, warnings, system information, and notifications (depending on the log-level setting).

The httpd.conf file contains the error log configuration for OHS. The logging mode is defined by the "OraLogMode" directive. The default is "odl-text", which produces the Oracle diagnostic logging format in a text file. Alternatively, change this to "odl-xml" to produce the Oracle diagnostic logging format in an XML file.

For Oracle diagnostic-style logging, "OraLogSeverity" directive is used for setting the log level.

For Apache-style logging, the ErrorLog and LogLevel directives identify the log file and the level of detail of the messages recorded. The default debug level is Warn.

Excessive logging can have some performance cost and may also fill disk space. The log level control should be used based on need. For requests that use dynamic resources, for example, requests that use `mod_osso` or `mod_plsql`, there is a performance cost associated with setting higher debugging levels, such as the debug level.

6.4 Advanced Tuning Considerations

This section provides advanced tuning recommendations which may or may not apply to your environment. Review the following recommendations to determine if the changes would improve your HTTP Server performance.

- Tuning Oracle HTTP Server
- Tuning Oracle HTTP Server Security

6.4.1 Tuning Oracle HTTP Server

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

- Analyzing Static Versus Dynamic Requests
- Managing PL/SQL Requests
- Limiting the Number of Enabled Modules
- Monitoring Oracle HTTP Server Performance

6.4.1.1 Analyzing Static Versus Dynamic Requests

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

Generally, you want to concentrate your tuning effort on dynamic pages because dynamic pages can be costly to generate. Also, by monitoring and tuning your application, you may find that much of the dynamically generated content, such as catalog data, can be cached, sparing significant resource usage.

6.4.1.2 Managing PL/SQL Requests

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

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

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

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

For example:

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

6.4.1.3 Limiting the Number of Enabled Modules

Oracle HTTP Server, which is now based on Apache 2.2, has a slight change in architecture in the way the requests are handled, compared to the previous release of Oracle HTTP Server, which was based on Apache 1.3.

In the new architecture, Oracle HTTP Server invokes the service function of each module that is loaded (in the order of definition in `httpd.conf` file) until the request is serviced. This indicates that there is some cost associated with invoking the service function of each module, to know if the service is accepted or declined.

Because of this change in architecture, consider placing the most frequently hit modules above the others in the `httpd.conf` file.

For the static page requests, which are directly deployed to Oracle HTTP Server and served by the default handler, the request has to go through all the modules before the default handler is invoked. This process can impact performance of the request so consider enabling only the modules that are required by the deployed application. Example, if "mod_plsql" is never used by the deployed application, disable it to maintain performance.

In addition, there are a few modules that register their hooks to do some work during the URL translation phase, which would add to the cost of request processing time. Example: mod_security, when enabled, has a cost of about 10% on CPU Cost per Transaction for the specweb benchmark. Again, enable only those modules that are required by your deployed applications to save CPU time.

6.4.1.4 Tuning the File Descriptor Limit

A lack of available file descriptors can cause a wide variety of symptoms which are not always easily traced back to the operating system’s file descriptor limit. Tuning the file descriptor limit can be accomplished by configuring the operating system’s hard limit for the user who starts the OHS. Once configured, the OHS will adjust the soft limit to match the operating system limit.

Configuring file descriptor limits is platform-specific. Refer to your operating system documentation for more information. The following code example shows the command for Linux:

```
APACHECTL_ULIMIT=ulimit -S -n `ulimit -H -n`
```

Note that this limit must be reconfigured after applying a patch set.

6.4.2 Tuning Oracle HTTP Server Security

This section covers the following topics:

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

6.4.2.1 Tuning Oracle HTTP Server Secure Sockets Layer (SSL)

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

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

This section covers the following SSL performance-related information:

- Section 6.4.2.1.1, "Caching SSL on Oracle HTTP Server"
- Section 6.4.2.1.2, "Using SSL Application Level Data Encryption"
- Section 6.4.2.1.3, "Tuning SSL Performance"

See Also: *Securing Applications with Oracle Platform Security Services*

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

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

The `SSLSessionCacheTimeout` directive in `ssl.conf` determines how long the server keeps a saved SSL session (the default is 300 seconds). Session state is discarded if it is not used after the specified time period, and any subsequent SSL request must establish a new SSL session and begin the handshake again. The `SSLSessionCache` directive specifies the location for saved SSL session information (the default location is the following directory):

```
$ORACLE_INSTANCE/diagnostics/logs/$COMPONENT_ TYPE/$COMPONENT_
NAME
```

Note that multiple Oracle HTTP Server processes can use a saved session cache file.

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

The reuse of saved SSL session state has some performance costs. When SSL session state is stored to disk, reuse of the saved state normally requires locating and retrieving the relevant state from disk. This cost can be reduced when using HTTP persistent connections. Oracle HTTP Server uses persistent HTTP connections by default, assuming they are supported on the client side. In HTTP over SSL as implemented by Oracle HTTP Server, SSL session state is kept in memory while the associated HTTP connection is persisted, a process which essentially eliminates the performance impacts associated with SSL session reuse (conceptually, the SSL connection is kept open along with the HTTP connection). For more information see Section 6.3.2, "Reducing Httpd Process Availability with Persistent Connections".

6.4.2.1.2 Using SSL Application Level Data Encryption In most applications using SSL, the data encryption cost is small compared with the cost of SSL session management. Encryption costs can be significant where the volume of encrypted data is large, and in such cases the data encryption algorithm and key size chosen for an SSL session can be significant. In general there is a trade-off between security level and performance.

Oracle HTTP Server negotiates a cipher suite with a client based on the SSLCipherSuite attribute specified in ssl.conf. OHS 11*g* uses 128 bit Encryption algorithm by default and no longer supports lower encryption. Note that the previous release [10.1.3x] used 64 bit encryption for Windows. For UNIX, the 10.x releases had 128 bit encryption used by default.

See Also: *Administering Oracle HTTP Server* for information on using supported cipher suites.

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

1. The SSL handshake is an inherently resource intensive process in terms of both CPU usage and response time. Thus, use SSL only where needed. Determine the parts of the application that require the security, and the level of security required, and protect only those parts at the requisite security level. Attempt to minimize the need for the SSL handshake by using SSL sparingly, and by reusing session state as much as possible. For example, if a page contains a small amount of sensitive data and several non-sensitive graphic images, use SSL to transfer the sensitive data only, use normal HTTP to transfer the images. If the application requires server authentication only, do not use client authentication. If the performance goals of an application cannot be met by this method alone, additional hardware may be required.
2. Design the application to use SSL efficiently. Group secure operations to take advantage of SSL session reuse and SSL connection reuse.
3. Use persistent connections, if possible, to minimize cost of SSL session reuse.

4. Tune the session cache timeout value (the `SSLSessionCacheTimeout` directive in `ssl.conf`). A trade-off exists between the cost of maintaining an SSL session cache and the cost of establishing a new SSL session. As a rule, any secured business process, or conceptual grouping of SSL exchanges, should be completed without incurring session creation more than once. The default value for the `SSLSessionCacheTimeout` attribute is 300 seconds. It is a good idea to test an application's usability to help tune this setting.
5. If large volumes of data are being protected through SSL, pay close attention to the cipher suite being used. The `SSLCipherSuite` directive specified in `ssl.conf` controls the cipher suite. If lower levels of security are acceptable, use a less-secure protocol using a smaller key size (this may improve performance significantly). Finally, test the application using each available cipher suite for the specified security level to find the optimal suite.
6. If SSL remains a bottleneck to the performance and scalability of your application, after taking the preceding considerations into account, consider deploying multiple Oracle HTTP Server instances over a hardware cluster or consider the use of SSL accelerator cards.

6.4.2.2 Tuning Oracle HTTP Server Port Tunneling

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

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

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

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

See Also: *Administering Oracle HTTP Server* for information on configuring OracleAS Port Tunneling

7 Oracle Metadata Service (MDS) Performance Tuning

This chapter provides tuning tips for Oracle Metadata Service (MDS).

- Section 7.1, "About Oracle Metadata Services (MDS)"
- Section 7.2, "Monitoring Oracle Metadata Service Performance"
- Section 7.3, "Basic Tuning Considerations"
- Section 7.4, "Advanced Tuning Considerations"

7.1 About Oracle Metadata Services (MDS)

Oracle Metadata Services (MDS) is an application server and Oracle relational database that keeps metadata in these areas: the ClassPath, the ServletContext, database repository and in some cases, the file system. One of the primary uses of MDS is to store customizations and persisted personalization for Oracle applications. MDS is used by components such as Oracle Application Development Framework (ADF) to manage metadata. Examples of metadata objects managed by MDS are: JSP pages and page fragments, ADF page definitions and task flows, and customized variants of those objects.

Note: Most of the Oracle Metadata Service configuration parameters are immutable and cannot be changed at run time unless otherwise specified.

7.2 Monitoring Oracle Metadata Service Performance

MDS uses DMS sensors to provide tuning and diagnostic information which can be viewed using Enterprise Manager. This information is useful, for example, to see if the MDS caches are large enough.

Information on DMS metrics can be found in the Fusion Middleware Control Console. Click **Help** at the top of the page to get more information. In most cases, the Help window displays a help topic about the current page. Click **Contents** in the Help window to browse the list of help topics, or click **Search** to search for a particular word or phrase.

7.3 Basic Tuning Considerations

Tuning is the adjustment of parameters to improve performance. The default MDS configuration must be tuned in almost all deployments. Please review the requirements and recommendations in this section carefully.

7.3.1 Tuning Database Repository

For optimal performance of MDS APIs, the database schema for the MDS repository must be monitored and tuned by the database administrator. This section lists some recommended actions to tune the database repository:

- Collecting Schema Statistics
- Increasing Redo Log Size
- Reclaiming Disk Space
- Monitoring the Database Performance

For additional information on tuning the database, see "Optimizing Instance Performance" in *Oracle Database Performance Tuning Guide*.

7.3.1.1 Collecting Schema Statistics

While MDS provides database indexes, they may not be used as expected due to a lack of schema statistics. If performance is an issue with MDS operations such as accessing or updating metadata in database repository, the database administrator must ensure that the statistics are available and current.

The following example shows one way that the Oracle database schema statistics can be collected:

```
execute dbms_stats.gather_schema_stats(ownname => '<username>',
estimate_percent => dbms_stats.auto_sample_size, method_opt=> 'for all
columns size auto', cascade=>true);
```

If the performance does not improve after statistics collection, then try to flush the database shared pool to clear out the existing SQL plans by using the following command:

```
alter system flush shared_pool;
```

In general, the database should be configured with automatic statistics recollection. For additional information on gathering statistics, see 'Automatic Performance Statistics" in *Oracle Database Performance Tuning Guide*.

7.3.1.2 Increasing Redo Log Size

The size of the redo log files can influence performance because the behavior of the database writer and archiver processes depend on the redo log sizes. Generally, larger redo log files provide better performance. Undersized log files increase checkpoint activity and can reduce performance.

For more information see "Sizing Redo Log Files" in *Oracle Database Performance Tuning Guide*.

7.3.1.3 Reclaiming Disk Space

While manual and auto-purge operations delete the metadata content from the repository, the database may not immediately reclaim the space held by tables and indexes. This may result in the disk space consumed by MDS schema growing.

Database administrators can manually rebuild the indexes and shrink the tables to increase performance and to reclaim disk space.

For more information see "Reclaiming Unused Space" in *Oracle Database Performance Tuning Guide*.

7.3.1.4 Monitoring the Database Performance

Database administrators must monitor the database (for example, by generating automatic workload repository (AWR) reports for Oracle database) to observe lock contention, I/O usage and take appropriate action to address the issues.

For more information see:

- "Generating Automatic Workload Repository Reports" in *Oracle Database Performance Tuning Guide*
- "Monitoring Performance" in *Oracle Database Administrator's Guide*.

7.3.2 Tuning Cache Configuration

MDS uses a cache to store metadata objects and related objects (such as XML content) in memory. MDS Cache is a shared cache that is accessible to all users of the application (on the same JVM). If a metadata object is requested repeatedly, with the same customizations, that object may be retrieved more quickly from the cache (a “warm” read). If the metadata object is not found in the cache (a “cold” read), then MDS may cache that object to facilitate subsequent read operations depending on the cache configuration, the type of metadata object and the frequency of access.

Cache can be configured or changed post deployment through MBeans. This element maps to the `MaximumCacheSize` attribute of the `MDSAppConfig` MBean. For more information see "Changing MDS Configuration Attributes for Deployed Applications" in *Administering Oracle Fusion Middleware*.

Note: MDS Metrics, visible in Enterprise Manager, are useful for tuning the MDS cache. In particular, "`IOs Per MO Content Get`" or "`IOs Per Metadata Object Get`" should be less than 1. If not, consider increasing the size of the MDS cache. For more information on viewing DMS metric information, see Section 7.2, "Monitoring Oracle Metadata Service Performance".

Having a correctly sized cache can significantly improve throughput for repeated reading of metadata objects. The optimal cache size depends on the number of metadata objects used and the individual sizes of these objects. Prior to packaging the Enterprise ARchive (EAR) file, you can manually update the cache-config in adf-config.xml, by adding the following entry:

```
<mds-config>
  <cache-config>
    <max-size-kb>200000</max-size-kb>
  </cache-config>
</mds-config>
```

Note: MDS cache grows in size as metadata objects are accessed until it hits `max-size-kb`. After that, objects are removed from the cache to make room as needed on a least recently used (LRU) basis to make room for new objects.

7.3.2.1 Enabling Document Cache

In addition to the main MDS cache, MDS uses a document cache in conjunction with each metadata store to store thumbnail information about metadata documents (base document and customization documents) in memory. The entry for each document is small (<100 bytes) and the cache size limit is specified in terms of the number of document entries. MDS calculates an appropriate default size limit for the document cache based on the configured maximum size of the MDS Cache, as follows:

- If MDS cache is disabled, MDS defaults to having no document cache.
- If MDS cache is enabled, MDS defaults the document cache size to one document entry per KB of document cache configured.
- If cache-config is not specified, MDS defaults to 10000 document entries.
- If MDS cache is set to a very small value, MDS uses a minimum size of 500 for document cache.

In general, the defaults should be sufficient in most cases. However, insufficient document cache size may impact performance. Prior to packaging the Enterprise ARchive (EAR) file, you can explicitly set document cache size by adding this entry to adf-config.xml:

```
<metadata-store-usage id="db1">
  <metadata-store …>
    <property name = …/>
  </metadata-store>
  <document-cache max-entries="10000"/>
</metadata-store-usage>
```

Note: Document cache is cleared when it exceeds the document-cache max-entries value. To avoid performance issues, consider increasing the document cache size if you receive a notification like the following for example:

```
NOTIFICATION: Document cache DBMetadataStore : MDS
Repository connection = <> exceeds its maximum
number of entries <NNNN>, so the cache is cleared.
```

The DMS metric "`IOs Per Document Get`" (visible in Enterprise Manager, see Section 7.2) should be less than 1. If not, consider increasing the document cache size.

7.3.3 Purging Document Version History

MDS keeps document version history in the database's metadata store. As version history accumulates, it requires more disk space and degrades read/write performance. Assuming the document versions are not part of an active label, there are two ways to purge version history:

- Auto Purge
- Manual Purge

Note: Purging version history manually may impact performance depending on the number of metadata updates that have been made since the last purge.

7.3.3.1 Auto Purge

The auto-purge interval can be configured or changed post deployment through MBeans. This element maps to the `AutoPurgeTimeToLive` attribute of the `MDSAppConfig` MBean. If your application uses the database store for MDS, you can set auto-purge by adding this entry in adf-config.xml prior to packaging the EAR:

```
<persistence-config>
  <auto-purge seconds-to-live="T"/>
</persistence-config>
```

In the example above, the auto-purge will be executed every *T* seconds and will remove versions that are older than the specified time *T* (in seconds). For more information, see "Changing MDS Configuration Attributes for Deployed Applications" in *Administering Oracle Fusion Middleware*.

7.3.3.2 Manual Purge

When you suspect that the database is running out of space or performance is becoming slower, you can manually purge existing version history using `WLST` command or through Oracle Enterprise Manager. Manual purging may impact performance, so plan to purge in a maintenance window or when the system is not busy.

For more information about manually purging version history, see "Purging Metadata Version History" in *Administering Oracle Fusion Middleware*.

7.3.4 Using Database Polling Interval for Change Detection

MDS employs a polling thread which queries the database to gauge if the data in the MDS in-memory cache is out of sync with data in the database. This can happen when metadata is updated in another JVM. If it is out of sync, MDS clears any out of date-cached data so subsequent operations see the latest versions of the metadata. MDS invalidates the document cache, as well as MDS cache, so subsequent operations have the latest version of the metadata.

The polling interval can be configured or changed post deployment through MBeans. The element maps to the `ExternalChangeDetection` and `ExternalChangeDetectionInterval` attributes of the `MDSAppConfig` MBean. Prior to packaging the Enterprise ARchive (EAR) file, you can configure the polling interval by adding this entry in adf-config.xml:

```
<mds-config>
  <persistence-config>
    <external-change-detection enabled="true" polling-interval-secs="T"/>
  </persistence-config>
</mds-config>
```

In the example above, 'T' specifies the polling interval in seconds. The minimum value is 1. Lower values cause metadata updates, that are made in other JVMs, to be seen more quickly. It is important to note, however, that a lower value can also create increased middle tier and database CPU consumption due to the frequent queries. By default, polling is enabled ('true') and the default value of 30 seconds should be suitable for most purposes. For more information, see "Changing MDS Configuration Attributes for Deployed Applications" in *Administering Oracle Fusion Middleware* ".

Note: When setting the polling interval, consider the following: if you poll too frequently, the database is queried for out-of-date versions; too infrequently, and those versions may stack up and polling can take longer to process.

7.4 Advanced Tuning Considerations

After you have performed the modifications recommended in the previous section, you can make additional changes that are specific to your deployment. Consider carefully whether the recommendations in this section are appropriate for your environment.

7.4.1 Analyzing Performance Impact from Customization

MDS customization may impact performance at run time.The impact from customization depends on many factors including:

- The type of customization that has been created (shared or user level)
- The percentage of metadata objects in the system which is customized. The lower this percentage the lower the impact of customization.
- The number of configured customization layers, and the efficiency of the customization classes.

There are two main types of customization:

- Shared Customizations: these are layers of customization corresponding to customization classes whose `getCacheHint` method returns `ALL_USERS` or `MULTI_USER`, meaning the layer applies to all or multiple users. Shared customizations are cached in the (shared) MDS cache.
- User Level Customizations (also known as Personalizations): these are layers of customization corresponding to customization classes whose `getCacheHint` method returns `SINGLE_USER`, meaning the layer applies to just one user. User customizations are generally cached on the user's session (HttpSession) until the user logs out.

For more information about customization concepts, writing customization classes, and configuring customization classes, see "Customizing Applications with MDS" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

Part III

Oracle Fusion Middleware Server Components

This part describes configuring Oracle Fusion Middleware server components to improve performance. It contains the following chapters:

8 Oracle Application Development Framework Performance Tuning

This chapter provides basic guidelines on how to maximize the performance and scalability of the Oracle Application Development Framework (ADF). This chapter covers design, configuration, and deployment performance considerations in the following sections:

- Section 8.1, "About Oracle ADF"
- Section 8.2, "Basic Tuning Considerations"
- Section 8.3, "Advanced Tuning Considerations"

This chapter assumes that you are familiar with building ADF applications. To learn about ADF, see the following guides:

- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Developing Web User Interfaces with Oracle ADF Faces*

8.1 About Oracle ADF

Oracle Application Development Framework (Oracle ADF) is an end-to-end application framework that builds on Java Platform, Enterprise Edition (Java EE) standards and open-source technologies to simplify and accelerate implementing service-oriented applications. Oracle ADF is suitable for enterprise developers who want to create applications that search, display, create, modify, and validate data using web, wireless, desktop, or web services interfaces. If you develop enterprise solutions that search, display, create, modify, and validate data using web, wireless, desktop, or web services interfaces, Oracle ADF can simplify your job. Used in tandem, Oracle JDeveloper 11g and Oracle ADF give you an environment that covers the full development lifecycle from design to deployment, with drag-and-drop data binding, visual UI design, and team development features built-in.

For more information see "Introduction to Oracle ADF" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

8.2 Basic Tuning Considerations

Before building, configuring, and deploying ADF applications, review the following tuning recommendations to achieve optimal performance:

- Oracle ADF Faces Configuration and Profiling
- Performance Considerations for ADF Faces

- Tuning ADF Faces Component Attributes
- Performance Considerations for Table and Tree Components
- Performance Considerations for autoSuggest
- Data Delivery - Lazy versus Immediate
- Performance Considerations for DVT Components

8.2.1 Oracle ADF Faces Configuration and Profiling

This section discusses the configuration and profiling concepts of the ADF Faces. Configuration options for Oracle ADF Faces are set in the `web.xml` file. Most of these have default values that are tuned for performance. Table 8–1 describes some of these configuration options.

Table 8–1 ADF Configuration Options

Parameter	Description
Compression View State `org.apache.myfaces.trinidad.COMPRESS_VIEW_STATE`	Controls whether or not the page state is compressed. Latency can be reduced if the size of the data is compressed. This parameter should be set to `True`.
Enhanced Debug `org.apache.myfaces.trinidad.resource.DEBUG`	Controls whether output should be enhanced for debugging or not. This parameter should be removed or set to `False`.
Check File Modification `oracle.adf.view.rich.CHECK_FILE_MODIFICATION`	Controls whether ADF faces check for modification date of JSP pages and discard any saved state if the file is changed. This parameter should be removed or set to `False`.
Client State Method `oracle.adf.view.rich.CLIENT_STATE_METHOD`	Specifies which type of saving (`all` or `token`) should be used when client-side state saving is enabled. The default value is `token`.
Client Side Log Level `oracle.adf.view.rich.LOGGER_LEVEL`	Sets the log level on the client side. The default value is `OFF`. This parameter should be removed or set to `False`.
Assertion Processing `oracle.adf.view.rich.ASSERT_ENABLED`	Specifies when to process assertions on the client side. The default value is `OFF`. This parameter should be removed or set to `False`.

Note: When you are profiling or measuring client response time using the Firefox browser, ensure that the Firebug plug-in is disabled. While this plug-in is very useful for getting information about the page and for debugging JavaScript code on the page, it can impact the total response time.

For more information on disabling the Firefox Firebug plug-in, see the Firefox Support Home Page at `http://support.mozilla.com/en-US/kb/`.

8.2.2 Performance Considerations for ADF Faces

Table 8–2 provides configuration recommendations that may improve performance of ADF Faces:

Table 8–2 Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Avoid inline JavaScript in pages.	Inline JavaScript can increase response payload size, will never be cached in browser, and can block browser rendering. Instead of using inline JavaScript, consider putting all scripts in .js files in JavaScript libraries and add scripts to the page using af:resource tag. **TIP**: Consider using af:resource rather than trh:script when possible.
Configure the JSP timeout parameter.	Using the JavaServer Pages (JSP) timeout parameter causes infrequently used pages to be flushed from the cache by the following setting in web.xml: `<servlet>` `<servlet-name>` `oraclejsp` `<init-param>` `<param-name>` `jsp_timeout` `</param-name>` `<param-value>` *`x`* `</param-value>` `</init-param>` `</servlet-name>` `</servlet>` NOTE: Set parameter *x* based on your own use case scenarios.
Create a single toolbar item with a drop-down popup.	When the browser size is small because of the screen resolution, the menubar/toolbar overflow logic becomes expensive in Internet Explorer 7 and 8. It especially has problems with laying out DOM structures with input fields. Create a single toolbar item with a drop-down popup and put all the input fields inside it. This popup should have deferred child creation and contentDelivery="lazy".
Remove unknown rowCount.	A table that has an unknown rowCount can impact performance because getting the last set of rows takes excessive scrolling from the user and the application can appear to be very slow. Remove unknown rowCount by setting DeferEstimatedRowCountProperty="false" on the view object (VO).
Disable pop-ups that cannot be displayed by the user.	The fnd:attachment component, when stamped in a table, can generate an excessive amount of DOM and client component. The amount of DOM + Client component is ~8K per cell which impacts the performance of the entire page especially on slower browsers. Most cells have no attachments initially and only one popup can be displayed by the user. Therefore, pop-ups that cannot be displayed by the user should have renderer="false". This will cut down the un-necessary DOM/client components sent to the browser. Similarly the DOM has a panelGroupLayout with a number of cells which are empty. There is no need to send DOM for empty cells.
Do not use hover pop-ups on navigation links.	A hover popup on a navigation link causes the navigation to wait for the hover to be fetched first. Consider removing the hover popup on the compensate workforce table navigation link column and, instead, place it on a separate column or on an icon inside the cell.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Increase table scrolling timeout.	Tables send a fetch request to the server on a scroll after a timeout. The timeout, before the fetch is sent to the server, is typically only 20ms if the user scrolls a short distance, but can increase to 200ms if the user scrolls further. Therefore performance can be impacted when the user scrolls to the bottom of a page and the table sends multiple requests to the server. To prevent the performance impact, consider increasing the timeout limit to 300ms.
Use a timeout to call _ prepareForIncompleteImages.	During Partial Page Rendering (PPR) some images may not load completely. When this occurs, the parent component must be notified that the size of one of its descendants has changed. In the past this was done by using the "complete" attribute on the image tag. Now with Internet Explorer 8 the complete attribute is always false to alleviate performance issues with Internet Explorer 7 and 8. The attribute shows as false even for cached images immediately after the PPR content is fetched. For Internet Explorer 8 use a timeout (10ms) to call _ prepareForIncompleteImages so that the image tag called right after the .xml HTTP request is processed. Note that this is not an issue for Mozilla Firefox or Google Chrome.
Cache the GetFirstVisibleRowKeyandRow.	Performance can be improved by locally caching the first visible Rowkey and row. This cached value can be deleted on a scroll or a resize.
Use partial page navigation.	Partial Page Navigation is a feature of the ADF Faces framework that enables navigating from one ADF Faces page to another without a full page transition in the browser.The new page is sent to the client using Partial Page Rendering (PPR)/Ajax channel. The main advantage of partial page navigation over traditional full page navigation is improved performance: the browser no longer re-interprets and re-executes Javascript libraries, and does not spend time for cleanup/initialization of the full page. The performance benefit from this optimization is very big; it should be enabled whenever possible. Some known limitations of this feature are: ▪ For the document's "metaContainer" facet (the HEAD section), only scripts are brought over with the new page. Any other content, such as icon links or style rules can be ignored. ▪ Applications cannot use anchor (hash) URLs for their own purposes.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Use page templates.	Page templates enable developers to build reusable, data-bound templates that can be used as a shell for any page. A developer can build one or more templates that provide structure and consistency for other developers building web pages. The templates have both static areas on them that cannot be changed when they are used and dynamic areas on them where the developer can place content specific to the page they are building. There are some important considerations when using templates: ▪ Since templates are present in every application page, they have to be optimized so that common performance impacts are avoided. Adding round corners to the template, for example, can impact the performance for every page. ▪ When building complex templates, sometimes it is easier to build them in multiple pieces and include them in the top-level template using `<f:subview>` tag. However, from a performance perspective, this is not typically recommended since it can impact memory usage on the server side. (`<f:subview>` introduces another level into the ID scoping hierarchy, which results in longer IDs. Long IDs have a negative impact on performance. Developers are advised to avoid using `<f:subview>` unless it is required. It is not necessary to use `<f:subview>` around `<jsp:include>` if you can ensure that all IDs are unique. For example, if you are using `<jsp:include>`, break a large page into multiple pieces for easier editing. And whenever possible, avoid using `<f:subview>`. If you are including content developed by someone else, use `<f:subview>` if you do not know which IDs the developer used. In addition, you do not have to put `<f:subview>` at the top of a region definition. ▪ Avoid long IDs in all cases, especially on pageTemplates, subviews, subforms, and on tables or within tables. Long IDs can have a performance impact on the server side, network traffic, and client processing.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Enable ADF rich client geometry management.	ADF Rich Client supports geometry management of the browser layout where parent components are in the UI explicitly. The children components are sized to stretch and fill up available space in the browser. While this feature makes the UI look better, it has a cost. The impact is on the client side where the browser must spend time resizing the components. The components that have geometry management by default are: PanelAccordion PanelStretchLayout PanelTabbed BreadCrumbs NavigationPane PanelSplitter Toolbar Toolbox Table Train **Notes:** ■ When using geometry management, try minimizing the number of child components that are under a parent geometry managed component. ■ The cost of geometry management is directly related to the complexity of child components. ■ The performance cost of geometry management can be smaller (as perceived by the user) for the pages with table or other data stamped components when table data streaming is used. The client-side geometry management can be executed while the browser is waiting for the data response from the server.
Use the ADF rich client overflow feature.	ADF Rich Client supports overflow feature. This feature moves the child components to the non-visible overflow area if they cannot fit the page. The components that have built-in support for overflow are: PanelTabbed, BreadCrumbs, NavigationPane, PanelAccordion, Toolbar, and Train. Toolbar should be contained in a Toolbox to handle the overflow. While there were several optimizations done to reduce the cost of overflow, it is necessary to pay special attention to the number of child components and complexity of each of them in the overflow component. Sometimes it is a good practice to set a big enough initial size of the overflow component such that overflow does not happen in most cases.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Use ADF Rich Client Partial Page Rendering (PPR).	ADF Rich Client is based on Asynchronous JavaScript and XML (Ajax) development technique. Ajax is a web development technique for creating interactive web applications, where web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, without the whole web page being reloaded. The effect is to improve a web page's interactivity, speed, and usability. With ADF Faces, the feature that delivers the Ajax partial page refresh behavior is called partial page rendering (PPR). PPR enables small areas of a page to be refreshed without having to redraw the entire page. For example, an output component can display what a user has chosen or entered in an input component or a command link or button can cause another component on the page to be refreshed. Two main Ajax patterns are implemented with partial page rendering (PPR): ■ native component refresh ■ cross-component refresh While the framework builds in native component refresh, cross-component refresh has to be done by developers in certain cases. Cross-component refresh is implemented declaratively or programmatically by the application developer defining which components are to trigger a partial update and which other components are to act as partial listeners, and so be updated. Using cross-component refresh and implementing it correctly is one of the best ways to improve client-side response time. While designing the UI page always think about what should happen when the use clicks a command button. Is it needed for the whole page to be refreshed or just an output text field? What should happen if the value in some field is updated? For more information, refer to *Developing Fusion Web Applications with Oracle Application Development Framework*). Consider a typical situation in which a page includes an `af:inputText` component, an `af:commandButton` component, and an `af:outputText` component. When the user enters a value for the `af:inputText`, then clicks the `af:commandButton`, the input value is reflected in the `af:outputText`. Without PPR, clicking the `af:commandButton` triggers a full-page refresh. Using PPR, you can limit the scale of the refresh to only those components you want to refresh, in this case the `af:outputText` component. To achieve this, you would do two things: ■ Set up the `af:commandButton` for partial submit by setting the `partialSubmit` attribute to `true`. Doing this causes the command component to start firing partial page requests each time it is clicked. ■ Define which components are to be refreshed when the partial submit takes place, in this example the `af:outputText` component, by setting the `partialTriggers` attribute for each of them to the id of the component triggering the refresh. In this example, this means setting the `partialTriggers` attribute of the `af:outputText` component to give the id of the `af:commandButton` component. The steps above achieve PPR using a command button to trigger the partial page refresh. The main reason why partial page rendering can significantly boost the performance is that full page refresh does not happen and the framework artifacts (such as ADF Rich Client JS library, and style sheets) are not reloaded and only a small part of page is refreshed. In several cases, this means no extra data is fetched or no geometry management. The ADF Rich Client has shown that partial page rendering results in the best client-side performance. Besides the impact on the client side, server-side processing can be faster and can have better server-side throughput and scalability.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Use ADF rich client navigation.	ADF Rich Client has an extensive support for navigation. One of the common use cases is tabbed navigation. This is currently supported by components like navigationPane which can bind to xmlMenuModel to easily define navigation. There is one drawback in this approach, however. It results in a full page refresh every time the user switches the tab. One option is to use panelTabbed instead. panelTabbed has built-in support for partial page rendering of the tabbed content without requiring any developer work. However, panelTabbed cannot bind to any navigational model and the content has to be available from within the page, so it has limited applicability.

Cache resources.	Developers are strongly encouraged to ensure that any resources that can be cached (images, CSS, JavaScript) have their cache headers specified appropriately. Also, client requests for missing resources on the server result in addition round trips to the server. To avoid this, make sure all the resources are present on the server.
	
	Consider using the ResourceServlet to configure web.xml to enable resource caching:
	
	`<servlet-mapping>`
	`     <servlet-name>resources</servlet-name>`
	`     <url-pattern>/js/*</url-pattern>`
	`   </servlet-mapping>`
	`<servlet-mapping>`
	`     <servlet-name>resources</servlet-name>`
	`     <url-pattern>/images/*</url-pattern>`
	`   </servlet-mapping>`

Reduce the size of state token cache.	This property is defined in web.xml org.apache.myfaces.trinidad.CLIENT_STATE_MAX_TOKENS in "token"-based client-side state saving and determines how many tokens should be preserved at any one time. The default value is 15. When this value is exceeded, state will be "forgotten" for the least recently viewed pages, which can impact users that actively use the Back button or that have multiple windows open simultaneously. In order to reduce live memory per session, consider reducing this value to 2. Reducing the state token cache to 2 means one Back button click is supported. For applications without support for a Back button, this value should be set to 1.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description

Define custom styles at the top of the page.	A common developer task is to define custom styles inside a regular page or template page. Since most browsers use progressive scanning of the page, a late introduction of styles forces the browser to recompute the page. This impacts the page layout performance. For better performance, define styles at the top of the page and possibly wrap them inside the ADF group tag.
	
	An HTML page basically has two parts, the "head" and the "body". When you put an `af:document` component on your page, this component creates both parts of the page for you. Any child component of the `af:document` is in the "body" part of the page. To get a component (or static CDATA content) to show up in the "head", use the "metaContainer" facet.
	
	To get a component (or static CDATA content) to display in the "head", use the "metaContainer" facet as follows:
	
	`<af:document title="#{attrs.documentTitle}" theme="dark">`
	`<f:facet name="metaContainer">`
	`<af:group><![CDATA[`
	`<style type="text/css">`
	`.TabletNavigationGlobal {`
	`text-align: right;`
	`padding-left: 0px;`
	`padding-right: 10px;`
	`white-space: nowrap;`
	`}`
	`HTML[dir=rtl] .TabletNavigationGlobal {`
	`text-align: left;`
	`padding-left: 10px;`
	`padding-right: 0px;`
	`}`
	`</style>`
	`]]>`
	`<af:facetRef facetName="metaContainer"/>`
	`</af:group>`
	`</f:facet>`
	`<af:form ...>`
	`<af:facetRef facetName="body"/>`
	`</af:form>`
	`</af:document>`
	
	If you use page templates, consider including `af:document` and `af:form` in the template definition and expose anything that you may want to customize in those tags through the page template attributes and page template `af:facetRef`. Your templates are then able to utilize the metaContainer facet if they have template-specific styling as shown above. Also, your usage pages do not have to repeat the same document and form tags on every page.
	
	See the *Developing Fusion Web Applications with Oracle Application Development Framework* for details about `af:facetRef`.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Optimize custom JavaScript code.	ADF Rich Client uses JavaScript on the client side. The framework itself provides most of the functionality needed. However, you may have to write custom JavaScript code. To get the best performance, consider bundling the JavaScript code into one JS lib (one JavaScript file) and deliver it to the client. The easiest approach is to use the ADF tag: `<af:resource type="javascript" source=" "/>`. If most pages require custom JavaScript code, the tag should be included in the application template. Otherwise, including it in particular pages can result in better performance. If custom the JavaScript code lib file becomes too big, then consider splitting it into meaningful pieces and include only the pieces needed by the page.Overall, this approach is faster since the browser cache is used and the html content of the page is smaller.
Disable debug output mode.	The `debug-output` element in the `trinidad-config.xml` file specifies whether output should be more verbose to help with debugging. When set to `TRUE`, the output debugging mechanism in Trinidad produces pretty-printed, commented HTML content. To improve performance by reducing the output size, you should disable the debug output mode in production environments. Set the `debug-output` element to `FALSE`, or if necessary, remove it completely from the `trinidad-config.xml` file.
Disable test automation.	Enabling test automation parameter `oracle.adf.view.rich.automation.ENABLED` generates a client component for every component on the page which can negatively impact performance. Set the `oracle.adf.view.rich.automation.ENABLED` parameter value to `FALSE` (the default value) in the `web.xml` file to improve performance.
Disable animation.	ADF Rich Client framework has client side animation enabled by default. Animation is introduced to provide an enhanced user experience. Some of the components, like popup table, have animation set for some of the operations. While using animation can improve the user experience, it can increase the response time when an action is executed. If speed is the biggest concern, then animation can be disabled by setting the flag in `trinidad-config.xml`
Disable client-side assertions.	Assertions on client-side code base can have a significant impact on client-side performance. Set the parameter value to `FALSE` (the default value) to disable client-side assertions. Also ensure that the `oracle.adf.view.rich.ASSERT_ENABLED` is not explicitly set to `TRUE` in the `web.xml` file.
Disable JavaScript Profiler.	When the JavaScript `oracle.adf.view.rich.profiler.ENABLED` profiler is enabled, an extra round-trip occurs on every page in order to fetch the profiler data. Disable the profiler in the `web.xml` file to avoid this extra round-trip.
Disable resource debug mode.	When resource debug mode is enabled, the HTTP response headers do not tell the browser that resources (JS libraries, CSS style sheets, or images) can be cached. Disable the `org.apache.myfaces.trinidad.resource.DEBUG` parameter in the `web.xml` file to ensure that caching is enabled.
Disable timestamp checking.	The `org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION` parameter controls whether jsp or jspx files are checked for modifications each time they are accessed. Ensure that the parameter value `org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION` is set to `FALSE` (the default value) in the web.xml file.

Table 8–2 (Cont.) Configuration Parameters for ADF Faces

Configuration Recommendation	Description
Disable checking for CSS file modifications.	The `org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION` parameter controls when CSS file modification checks are made. To aid in performance, this configuration option defaults to `false` - do not check for css file modifications. Set this to `TRUE` if you want the skinning css file changes to be reflected without stopping or starting the server.
Enable content compression.	By default, style classes that are rendered are compressed to reduce page size. In production environments, make sure you remove the `DISABLE_CONTENT_COMPRESSION` parameter from the `web.xml` file or set it to `FALSE`. For debugging, turn off the style class content compression. You can do this by setting the `DISABLE_CONTENT_COMPRESSION` property to `TRUE`.
Enable JavaScript obfuscation.	ADF Faces supports a run time option for providing a non-obfuscated version of the JavaScript library. The obfuscated version is supplied by default, but the non-obfuscated version is supplied for development builds. Obfuscation reduces the overall size of the JavaScript library by about 50%. To provide an obfuscated ADF Faces build, set the `org.apache.myfaces.trinidad.DEBUG_JAVASCRIPT` parameter to `FALSE` in the web.xml file. There are two ways to check that the code is obfuscated using Firefox with Firebug enabled: Check the download size: **1.** Ensure that "All" or "JS" is selected on the Net tab. **2.** Locate the "`all-11-version.js`" entry. **3.** Check the size of the column. It should be about 1.3 MB (as opposed to 2.8 MB). Check the source: **1.** From the Script tab select "all-11-*version*.js from the drop-down menu located above the tabs. **2.** Examine the code. If there are comments and long variable names, the library is not obfuscated. Note: Copyright comments are kept even in the obfuscated version of the JS files.
Enable library partitioning.	In the Oracle 11*g* Release, library partitioning is on by default. In previous versions library partitioning was off by default. Ensure that the library partitioning is on by validating the `oracle.adf.view.rich.libraryPartitioning.DISABLED` property is set to `false` in the web.xml file.

8.2.3 Tuning ADF Faces Component Attributes

Table 8–3 provides configuration recommendations for ADF Faces Component Attributes:

Table 8–3 ADF Faces Component Attributes

Configuration Recommendation	Description
Use the "immediate" attribute.	ADF Rich Client components have an `immediate` attribute. If a component has its `immediate` attribute set to `TRUE` (`immediate="true"`), then the validation, conversion, and events associated with the component are processed during the applyRequestValues phase. These are some cases where setting `immediate` to `TRUE` can lead to better performance. ■ The commandNavigationItem in the navigationPane can use the `immediate` attribute set to `TRUE` to avoid processing the data from the current screen while navigating to the new page. ■ If the input component value has to be validated before the other values, `immediate` should be set to `TRUE`. In case of an error it be detected earlier in the cycle and additional processing be avoided. ADF Rich Client is built on top of JSF and uses standard JSF lifecycle. See "Understanding the JSF and ADF Faces Lifecycles" in *Developing Web User Interfaces with Oracle ADF Faces*. There are some important issues associated with the `immediate` attribute. Refer to "Using the Immediate Attribute" in *Developing Web User Interfaces with Oracle ADF Faces* for more information. Note that this is an advanced feature. Most of the performance improvements can be achieved using the `af:subform` component. Refer to *Developing Web User Interfaces with Oracle ADF Faces* for `af:subform` details.
Use the "visible" and "rendered" attributes.	All ADF Faces Rich Client display components have two properties that dictate how the component is displayed on the page: ■ The `visible` property specifies simply whether the component is to be displayed on the page, or is to be hidden. ■ The `rendered` property specifies whether the component shall exist in the client page at all. The EL expression is commonly used to control these properties. For better performance, consider setting the component to not rendered instead of not visible, assuming there is no client interaction with the component. Making a component not rendered can improve server performance and client response time since the component does not have client side representation.
Use client-side events.	ADF Rich Client framework provides the client-side event model based on component-level events rather than DOM level. The client-side event model is a very useful feature that can speed up the application. Review the following performance considerations: ■ Consider using client-side events for relatively simple event handling that can be done on the client side. This improves client side performance by reducing the number of server round trips. Also, it can increase server-side throughput and scalability since requests do not have to be handled by the server. ■ By default, the events generated on the client by the client components are propagated to the server. If a client-side event handler is provided, consider canceling the event at the end of processing so that the event does not propagate to the server.

Table 8–3 (Cont.) ADF Faces Component Attributes

Configuration Recommendation	Description
Use the "id" attribute.	The "id" attribute should not be longer than 7 characters in length. This is particularly important for naming containers. A long id can impact performance as the amount of HTML that must be sent down to the client is impacted by the length of the ids.
Use client-side components.	ADF Rich Client framework has client-side components that play a role in client-side event handling and component behavior. The `clientComponent` attribute is used to configure when (or if) a client-side component should be generated. Setting `clientComponent` attribute to `TRUE` has a performance impact, so determine if its necessary to generate client-side components. For more information, see "Client-side Components" in *Developing Web User Interfaces with Oracle ADF Faces*.
Set the `childCreation` attribute on `af:popup` to `deferred` for a server-side performance enhancement	Setting `childCreation` to `deferred` postpones construction of the components under the popup until the content is delivered. A deferred setting can therefore reduce the footprint of server-side state in some cases. CAUTION: This approach CANNOT be used if any of the following tags are present inside the popup: ■ f:attribute ■ af:setPropertyListener ■ af:clientListener ■ af:serverListener It also CANNOT be used if you need to refer to any child components of the popup before the popup is displayed. Setting childCreation="deferred" will postpone creating any child components of the popup and you cannot refer to them until after the popup is shown.

8.2.4 Performance Considerations for Table and Tree Components

Table, Tree, and TreeTable are some of the most complex, and frequently used, components. Since these components can include large sets of data, they can be the common source of performance problems. Table 8–4 provides some performance recommendations.

Table 8–4 Table and Tree Component Configurations

Configuration Recommendation	Description
Use editingMode="clickToEdit".	When using editingMode="editAll" all content of the editable values holders and their client components is sent. This can significantly increase the HTTP payload and the Document Object Model (DOM) content on the client. Consider switching to editingMode="clickToEdit" to reduce the amount of transmitted data and potentially improve user interaction.
Reduce fetchSize when possible.	A larger fetch size attribute on af:table implies that more data needs to be processed, fetched from the server, and displayed on the client. This can also increase the amount of DOM displayed on the client.
Modify table fetch size.	Tables have a fetch size which defines the number of rows to be sent to the client in one round-trip. To get the best performance, keep this number low while still allowing enough rows to fulfill the initial table view port. This ensures the best performance while eliminating extra server requests. In addition, consider keeping the table fetch size and iterator range size in sync. By default, the table fetch size is set to the EL expression `#{bindings.<name>.rangeSize}` and should be equal to the iterator size. For more information see "Using Tables and Trees" in *Developing Web User Interfaces with Oracle ADF Faces*.
Disable column stretching.	Columns in the table and treeTable components can be stretched so that there is no unused space between the end of the last column and the edge of the table or treeTable component. This feature is turned off by default due to potential performance impacts. Turning this feature on may have a performance impact on the client rendering time, so use caution when enabling this feature with complex tables.
Consider using header rows and frozen columns only when necessary.	The table component provides features that enable you to set the row Header and frozen columns. These options can provide a well-designed interface which can lead to a good user experience. However, they can impact client-side performance. To get the best performance for table components, use these options only when they are needed.
Consider using visitTree instead of invokeOnComponent.	A partial visit using visitTree is always at least as fast as invokeOnComponent. In addition, for components controlling visiting, providing both invokeOnComponent and visitTree implementations is a source of errors. Consider deprecating invokeOnComponent and use visitTree instead. For more information see "Using Tables and Trees" in *Developing Web User Interfaces with Oracle ADF Faces*.

8.2.5 Performance Considerations for autoSuggest

`autoSuggest` is a feature that can be enabled for inputText, inputListOfValues, and inputComboboxListOfValues components. When the user types characters in the input field, the component displays a list of suggested items. The feature performs a query in the database table to filter the results. In order to speed up database processing, a database index should be created on the column for which autosuggest is enabled. This improves the component's response times especially when the database table has a large number of rows.

8.2.6 Data Delivery - Lazy versus Immediate

Data for Table, Tree, and other stamped components can be delivered immediately or lazily. By default, lazy delivery is used. This means that data is not delivered in the initial response from the server. Rather, after the initial page is rendered, the client asks the server for the data and gets it as a response to the second request.

In the case of immediate delivery, data can be in line with the response to the page request. It is important to note that data delivery is per component and not per page. This means that these two can be mixed on the same page.

When choosing between these two options, consider the following:

Lazy Delivery (default)	Lazy delivery should be used for tables, or other stamped components, which are known to have slow fetch time. The examples are stamped components are the ones based on data controls using web services calls or other data controls with slow data fetch. Lazy delivery can also be used on pages where content is not immediately visible unless the user scrolls down to it. In this case the time to deliver the visible context to the client will be shorter, and the user perceives better performance. Lazy delivery is implemented using data streaming technique. The advantage of this approach is that the server has the ability to execute data fetches in parallel and stream data back to the client as soon as the data is available. The technique performs very well for a page with two tables, one that returns data very quickly and one that returns data very slowly. Users see the data for the fast table as soon as the data is available. Executing data fetches in parallel also speeds up the total time to fetch data. This gives an advantage to lazy loading in cases of multiple, and possibly slow, data fetches. While streaming is the default mechanisms to deliver data in lazy mode, parallel execution of data controls is not. In order to enable parallel execution, open the page definition and change `RenderHint` on the iterator to *background*. In certain situations, the advantage of parallel execution is faster response time. Parallel execution could potentially use more resources due to multiple threads executing request in parallel and possibly more database connections will be opened. Consider using parallel execution only when there are multiple slow components on the page and the stamped components belong to different data control frames (such as isolated taskflows). Since parallel execution synchronizes on the data control frame level, when there is a single data control frame parallel execution may not improve performance.
Immediate Delivery	Immediate delivery (`contentDelivery="immediate"`) should be used if table data control is fast, or if it returns a small set of data. In these cases the response time be faster than using lazy delivery. Another advantage of immediate delivery is less server resource usage, compared to lazy delivery. Immediate delivery sends only one request to the server, which results in lower CPU and memory usage on the server for the given user interaction.

8.2.7 Performance Considerations for DVT Components

DVT components are data visualization components built on top of ADF Rich Client components. DVT components include graphs, gauges, Gantt charts, pivot tables and maps. Table 8–5 provides some configuration recommendations for DVT components:

Table 8–5 DVT Component Configurations

Configuration Recommendation	Description
Modify the RangeSize attribute.	The RangeSize attribute defines the number of rows to return simultaneously. A RangeSize value of -1 causes the iterator to return all the rows. Using a lower value may improve performance, but it may be harder to stop the data and any data beyond rangeSize is not available in the view.
Use horizontal text instead of vertical text.	By default, pivot tables use horizontal text for column headers. However, there is an option to use vertical text as well. Vertical text can be used by specifying a CSS style for the header format such as: `writing-mode:tb-rl;filter:flipV flipH;` While vertical text can look better in some cases, it has a performance impact when the Firefox browser is used. The problem is that vertical text is not native in Firefox as it is in Internet Explorer. To show vertical text, the pivot table uses images produced by GaugeServlet. These images cannot be cached as the text is dynamic and depends on the binding value. Due to this, every rendering of the pivot table incurs extra round-trips to the server to fetch the images, which impact network traffic, server memory, and CPU. To have the best performance, consider using horizontal text instead of vertical text.

8.3 Advanced Tuning Considerations

After you have performed the tuning modifications recommended in the previous section, you can make additional changes that are specific to your ADF Server deployment. Consider carefully whether the recommendations in this section are appropriate for your environment.

8.3.1 ADF Server Performance

Oracle ADF Server components consist of the non-UI components within ADF. These include the ADF implementations of the model layer (ADFm), business services layer (ADFbc), and controller layer (ADFc). As the server components are highly configurable, it is important to choose the combination of configurations that best suits the available resources with the specified application performance and functionality.

Note: When using ADFm, consider using deferred execution and monitor the refresh conditions to maintain performance.

8.3.1.1 HTTP Session Timeout Tuning

For ADF applications with a significant user community, the amount of memory held by sessions waiting to expire can negatively impact performance when the default HTTP session timeout of 45 minutes is used. The memory being held can be higher than what is physically available, causing the server to not be able to handle the load. For large numbers of users, such as those using a public facing website, the session timeout should be as short as possible.

To improve performance, consider modifying the default session timeout value (in minutes) in the `web.xml` file. Use a session timeout value that works with your use case scenario. The example below shows a session timeout of 10 minutes:

```
<session-config>
 <session-timeout>
 10
 </session-timeout>
</session-config>
```

8.3.1.2 View Objects Tuning

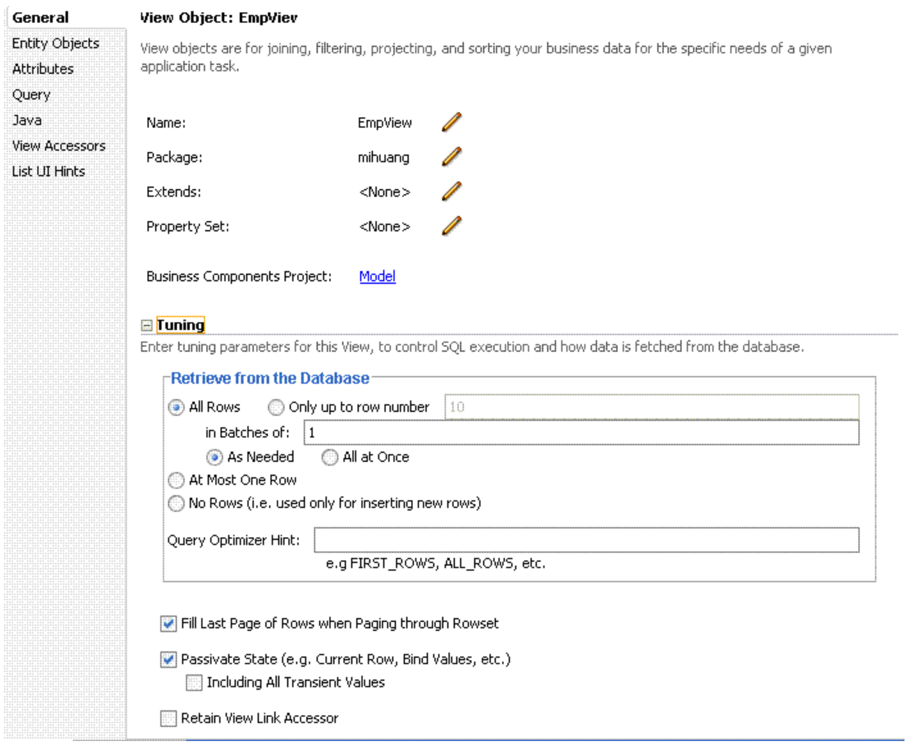
View objects (VOs) provide many tuning options to enable a developer to tailor the View Object to the application's specific needs. View Objects should be configured to use the minimal feature set required to fulfill the functional requirement. The *Developing Fusion Web Applications with Oracle Application Development Framework* provides detailed information on tuning View Objects. Provided here are some tips pertaining to View Object performance.

8.3.1.2.1 Creating View Objects To maximize View Object performance, the View Object should match the intended usage. For instance, data retrieved for a list of values pick-list is typically read-only, so a read-only View Object should be used to query this data. Tailoring the View Object to the specific needs of the application can improve performance, memory usage, CPU usage, and network usage.

View Object Type	Description
Read-only View Objects	Consider using a read-only View Object if the View Object does not have to insert or update data. There are two options for read-only View Objects: ▪ Non-updatable EO-based View Objects ▪ Expert-mode View Objects Non-updatable EO-based View Objects offer the advantage of a customizable select list at run time which retrieve attributes needed in the UI, data reads from local cache (instead of re-executing a database query), and data consistency with other updatable View Objects based on the same EO. Expert-mode View Objects have the ability to perform SQL operations not supported by EOs and avoid the small performance impact from coordinating View Object and EO rows. EO-based View Objects can be marked non-updatable by deselecting the "updatable" option in the selected EO for the View Object, which can also be done by adding the parameter `ReadOnly="true"` on the `EntityUsage` attribute in the View Object XML definition.
Insert-only View Objects	For View Objects that are used only for inserting records, you can prevent unnecessary select queries from being executed when using the View Object. To do this, set the option No Rows in the Retrieve from the Database group box in the View Objects Overview tab. This sets MaxFetchSize to 0 (zero) for the View Object definition.
run time-created View Objects	View Objects can be created at run time using the `createViewObjectFromQueryStmt()` API on the AM. However, avoid using run time-created View Objects unless absolutely necessary due to potential performance impacts and complexity of tuning.

8.3.1.2.2 Configuring View Object Data Fetching View Object performance is largely dependent on how the view object is configured to fetch data. If the fetch options are not tuned correctly for the application, then the view object may fetch an excessive amount of data or may take too many round-trips to the database. Fetch options can be configured through the **Retrieve from the Database** group box in the View Object dialog Figure 8–1.

Figure 8–1 View Object Dialog



View Object: EmpView

View objects are for joining, filtering, projecting, and sorting your business data for the specific needs of a given application task.

Name:	EmpView
Package:	mihuang
Extends:	<None>
Property Set:	<None>
Business Components Project:	Model

Tuning

Enter tuning parameters for this View, to control SQL execution and how data is fetched from the database.

Retrieve from the Database

[✓ All Rows] Only up to row number 10

in Batches of: 1

[✓ As Needed] All at Once

At Most One Row

No Rows (i.e. used only for inserting new rows)

Query Optimizer Hint:

e.g FIRST_ROWS, ALL_ROWS, etc.

[✓] Fill Last Page of Rows when Paging through Rowset

[✓] Passivate State (e.g. Current Row, Bind Values, etc.)

[ ] Including All Transient Values

[ ] Retain View Link Accessor

Fetch Option	Description
Fetch Mode	The default fetch option is the All Rows option, which is retrieved as needed (`FetchMode="FETCH_AS_NEEDED"`) or all at once (`FetchMode="FETCH_ALL"`), depending on which option is appropriate. The As Needed option ensures that an `executeQuery()` operation on the view object initially retrieves only as many rows as necessary to fill the first page of a display. The number of rows is set based on the view object's range size.
Fetch Size	In conjunction with the fetch mode option, the Batches field controls the number of records fetched simultaneously from the database (FetchSize in the View Object, XML). The default value is 1, which may impact performance unless only 1 row is fetched. The suggested configuration is to set this value to *n*+1 where *n* is the number of rows to be displayed in the user interface. Note that for DVT objects, Fetch Size should be *n*+1 where *n* is either rangeSize or the likely maximum rowset size if rangeSize is -1.
Max Fetch Size	The default max fetch size for a View Object is -1, which means that there is no limit to the number of rows the View Object can fetch. Setting a max fetch size of 0 (zero) makes the View Object insert-only. In cases where the result set should only contain *n* rows of data, the option Only Up to Row Number should be selected and set or call `setMaxFetchSize(N)` to set this programmatically. To set this manually, add the parameter `MaxFetchSize` to the View Object XML. For View Objects whose WHERE clause expects to retrieve a single row, set the option At Most One Row. This option ensures that the view object knows not to expect any more rows and skips its normal test for that situation. In this case no select query is issued and no rows are fetched. Max fetch size can also be used to limit the impact from an non-selective query that may return hundreds (or thousands) of rows. In such cases, specifying the max fetch size limits the number of rows that can be fetched and stored into memory.

Fetch Option	Description
Forward-Only Mode	If a data set is only traversed going forward, then forward-only mode can help performance when iterating through the data set. This can be configured by programmatically calling `setForwardOnly(true)` on the View Object. Setting forward-only can also prevent caching previous sets of rows as the data set is traversed.

8.3.1.2.3 Additional View Object Configurations

Table 8–6 provides additional tuning considerations when using the View Object:

Table 8–6 Additional View Object Configurations

Configuration Recommendation	Description
Optimize large data sets.	View Objects provide a mechanism to page through large data sets so that a user can jump to a specific page in the results. This is configured by calling `setRangeSize(N)` followed by `setAccessMode(RowSet.RANGE_PAGING)` on the View Object where N is the number of rows contained within 1 page. When navigating to a specific page in the data set, the application can call `scrollToRangePage(P)` on the View Object to navigate to page P. Range paging fetches and caches only the current page of rows in the View Object row cache at the cost of another query execution to retrieve each page of data. Range paging is not appropriate where it is beneficial to have all fetched rows in the View Object row cache (for example, when the application must read all rows in a data set for an LOV or page back and forth in records of a small data set).
Disable "spillover" configurations when possible.	You can use the data source as "virtual memory" when the JVM container runs out of memory. By default this is disabled and can be enabled (if needed) by setting `jbo.use.pers.coll=true`. Keep this option disabled (if possible) to avoid a potential performance impact.
Review SQL style configuration.	If the generic SQL92 SQL style is used to connect to generic SQL92-compliant database, then some View Object tuning options do not apply. The View Object fetch size is one such tuning option. When SQL92 SQL style is used, the fetch size defaults to 10 rows, regardless of what is configured for the View Object. The SQL style is set when defining the database connection. By default when defining an Oracle database connection, the SQL style can be `Oracle`. To manually override the SQL style, pass the parameter `-Djbo.SQLBuilder="SQL92"` to the JVM at startup.
Use bind variables for view object queries.	If the query associated with the View Object contains values that may change from execution to execution, consider using bind variables. This may help to avoid re-parsing the query on the database. Bind variables can be added to the View Object in the Query section of the View Object definition.
Use query optimizer hints for view object queries.	The View Object can pass hints to the database to influence which execution plan to use for the associated query. The optimizer hints can be specified in the Retrieve from the Database group box.
Use dynamic SQL generation.	View Objects can be configured to dynamically generate SQL statements at run time instead of defining the SQL at design time. A View Object instance, configured with generating SQL statements dynamically, can avoid re-querying a database. This is especially true during page navigation if a subset of all attributes with the same key Entity Object list is used in the subsequent page navigation. Performance can be improved by activating a superset of all the required attributes to eliminate a subsequent query execution.

8.3.1.3 Batch Processing

Batch processing enables multiple inserts, updates, and deletes to be processed together when sending the operations to the database. Enabling this feature is done on the Entity Object (EO) by either selecting the "Use Update Batching" check box in the

Tuning section of the EO's General tab, or by directly modifying the EO's XML file and adding the parameter `BatchThreshold` with the specified batch size to the `Entity` attribute.

The `BatchThreshold` value is the threshold at which a group of operations can be batched instead of performing each operation one at a time. If the threshold is not exceeded, then rows may be affected one at a time. On the other hand, more rows than specified by the threshold can be batched into a single batch.

Note that the `BatchThreshold` configuration for the EO is not compatible if an attribute in the EO exists with the configuration to refresh after insert (`RetrievedOnInsert="true"`) or update (`RetrievedOnUpdate="true"`).

8.3.1.4 RangeSize Tuning

This parameter controls the number of records ADFm requests from the BC layer simultaneously. The default `RangeSize` is 25 records. Consider setting this value to the number of records to be displayed in the UI simultaneously for the View Object so that the number of round-trips between the model and BC layers is reduced to one. This is configured in the `Iterator` attribute of the corresponding page's page definition XML.

8.3.1.5 Application Module Design Considerations

Designing an application's module granularity is an important consideration that can significantly impact performance and scalability. It is important to note that each root application module generally holds its own database connection. If a user session consumes multiple root application modules, then that user session can potentially hold multiple database connections simultaneously. This can occur even if the connections are not actively being used, due to the general affinity maintained between an application module and a user session. To reduce the possibility that a user can hold multiple connections at once, consider the following options:

- Design larger application modules to encompass all of the functionality that a user needs.
- Nest smaller application modules under a single root application module so that the same database connection can be shared among the nested application modules.
- Use lazy loading for application modules. In the Application Module tuning section, customize runtime instantiation behavior to use lazy loading. Lazy loading can also be set JVM-wide by adding the following JVM argument:

  ```
  -Djbo.load.components.lazily=true
  ```

8.3.1.6 Application Module Pooling

Application module (AM) pooling enables multiple users to share several application module instances. The configurations for the AM pool vary depending on the expected usage of the application.

Most of the AM pool parameters can be set through Oracle JDeveloper. The configurations are saved in `bc4j.xcfg`, which can be manually edited if needed. Parameters can also be set at the system level by specifying these as JVM parameters (`-D`property=value). The `bc4j.xcfg` configuration takes precedence over the JVM configuration; this enables a generic system-level configuration to be overridden by an application-specific exception.

Table 8–7 Application Module (AM) Pool Tuning

Configuration Recommendation	Description
Optimize the number of AM pools in the application.	Parameters applied at the system level are applied per AM pool. If the application uses more than 1 AM pool, then system-level values for the number of AM instances must be multiplied by the number of AM pools to realize the actual limits specified on the system as a whole. For example, if an application uses 4 separate AM pools to service the application, and a system-level configuration is used to limit the max AM pool size to 100, then this can result in a maximum of 400 AM instances (4 pools * 100 max pool size). If the intent is to limit the entire application to a max pool size of 100, then the system-level configuration should specify a max pool size of 25 (100 max pool size / 4 pools). Finer granularity for configuring each AM pool can be achieved by configuring each pool separately through JDev or directly in bc4j.xcfg.
Optimize the number of database connections.	By default AM instances retain their database connections even when checked back into the AM pool. There are many performance benefits to maintain this association. To maintain performance, consider configuring more AM instances than the maximum number of specified database connections. NOTE: If you have an AM pool that needs to be used as root pool, consider tuning at the specific AM pool level. For pools that are infrequently used, consider tuning pool sizes on the pool level so that top-level application parameters are not used.

8.3.1.6.1 General AM Pool Configurations The following guidelines can be used as a general starting point when tuning AM and AM pool behavior. More specific tuning for memory or CPU usage can be found in Section 8.3.1.6.2, "Configuring AM Pool Sizing".

Table 8–8 AM Pool Tuning Parameters

Parameter	Description
Initial Pool Size `jbo.ampool.initpoolsize`	Specifies the number of application module instances to create when the pool is initialized (default is zero). Setting a nonzero initial pool size increases the time to initialize the application, but improves subsequent performance for operations requiring an AM instance. Configure this value to 10% more than the anticipated number of concurrent AM instances required to service all users.
Maximum Pool Size `jbo.ampool.maxpoolsize`	Specifies the maximum number of application module instances that the pool can allocate (default is 4096). The pool can never create more application module instances than this limit imposes. A general guideline is to configure this to 20% more than the initial pool size to allow for some additional growth.
Minimum Available Size `jbo.ampool.minavailablesize`	The minimum number of available application module instances that the pool monitor should leave in the pool during a resource cleanup operation, when the server is under light load. Set to 0 (zero) if you want the pool to shrink to contain no instances when all instances have been idle for longer than the idle time-out after a resource cleanup. The default is 5 instances. While application module pool tuning allows different values for the `jbo.ampool.minavailablesize` | `jbo.ampool.maxavailablesize` parameters, in most cases it is fine to set these minimum and maximum tuning properties to the same value.

Table 8–8 (Cont.) AM Pool Tuning Parameters

Parameter	Description
Maximum Available Size `jbo.ampool.maxavailablesize`	The ideal maximum number of available application module instances in the pool when the server is under load. When the pool monitor wakes up to do resource cleanup, it will try to remove available application module instances to bring the total number of available instances down to this ideal maximum. Instances that have been not been used for a period longer than the idle instance time-out will always get cleaned up at this time, then additional available instances will be removed if necessary to bring the number of available instances down to this size. The default maximum available size is 25 instances. Configure this to leave the maximum number of available instances desired after a resource cleanup. A lower value generally results in more application module instances being removed from the pool on a cleanup. While application module pool tuning allows different values for the `jbo.ampool.maxavailablesize` | `jbo.ampool.minavailablesize` parameters, in most cases it is fine to set these minimum and maximum tuning properties to the same value.
Referenced Pool Size `jbo.recyclethreshold`	Specifies the maximum number of application module instances in the pool that attempt to preserve session affinity for the next request made by the session that used them last before releasing them to the pool in managed-state mode (default is 10). The referenced pool size should always be less than or equal to the maximum pool size. This enables the configured number of available instances to try and remain "loyal" to the affinity they have with the most recent session that released them in managed state mode. Configure this value to the expected number of concurrent users that perform multiple operations with short think times. If there are no users expected to use the application with short think times, then this can be configured to 0 (zero) to eliminate affinity.
Maximum Instance Time to Live `jbo.ampool.timetolive`	The number of milliseconds after which to consider an connection instance in the pool as a candidate for removal during the next resource cleanup regardless of whether it would bring the number of instances in the pool below minavailablesize. The default is 3600000 milliseconds of total time to live (which is 3600 seconds, or one hour). A lower value reduces the time an application module instance can exist before it must be removed at the next resource cleanup. The default value is sufficient for most applications. A higher value increases the time an application module instance can exist before it must be removed at the next cleanup.
Idle Instance Timeout `jbo.ampool.maxinactiveage`	The number of milliseconds after which to consider an inactive application module instance in the pool as a candidate for removal during the next resource cleanup. The default is 600000 milliseconds of idle time (which is 600 seconds, or ten minutes). A lower value results in more application module instances being marked as a candidate for removal at the next resource cleanup. A higher value results in fewer application module instances being marked as a candidate for removal at the next resource cleanup.

Table 8–8 (Cont.) AM Pool Tuning Parameters

Parameter	**Description**
Pool Polling Interval `jbo.ampool.monitorsleepinterval`	The length of time in milliseconds between pool resource cleanup. While the number of application module instances in the pool will never exceed the maximum pool size, available instances which are candidates for getting removed from the pool do not get "cleaned up" until the next time the application module pool monitor wakes up to do its job. The default is to have the application module pool monitor wake up every 600000 milliseconds (which is 600 seconds, or ten minutes). Configuring a lower interval results in inactive application module instances being removed more frequently to save memory. Configuring a higher interval results in less frequent resource cleanups.
Failover `jbo.dofailover`	Specifies whether to disable or enable failover. By default, failover is disabled. To enable failover, set the parameter to `true`. **NOTE**: When enabling application module state passivation, a failure can occur when Oracle WebLogic Server is configured to forcibly release connection back into the pool. A failure of this type produces a SQLException (Connection has already been closed) that is saved to the server log. The exception is not reported through the user interface. To ensure that state passivation occurs and changes are saved, set an appropriate value for the weblogic-application.xml deployment descriptor parameter `inactive-connection-timeout-seconds` on the `<connection-check-params>` pool-params element. Setting the deployment descriptor parameter to several minutes, in most cases, should avoid forcing the inactive connection timeout and the resulting passivation failure. Adjust the setting as needed for your environment.
Locking Mode `jbo.locking.mode`	Specifies the locking mode (`optimistic` or `pessimistic`). The default is `pessimistic`, which means that a pending transaction state can be created on the database with row-level locks. With pessimistic locking mode, each time an AM is recycled, a rollback is issued in the JDBC connection. Web applications should set the locking mode to `optimistic` to avoid creating the row-level locks.
Database Connection Pooling `jbo.doconnectionpooling`	Specifies whether the AM instance can be disconnected from the database connection when the AM instance is returned to the AM pool. This enables an application to size the AM pool larger than the database connection pool. The default is `false`, which means that an AM instance can retain its database connection when the AM instance is returned to the AM pool. When set to `true`, the AM can release the database connection back to the database connection pool when the AM instance is returned to the AM pool. Note that before an AM is disconnected from the database connection, a rollback can be issued on that database connection to revert any pending database state.
Transaction Disconnect Level `jbo.txn.disconnect_level`	When used in conjunction with `jbo.doconnectionpooling=true`, specifies BC4J behavior for maintaining JDBC ResultSets. By default `jbo.txn.disconnect_level` is 0, and passivation can be used to close any open ResultSets when the database connection is disconnected from the AM instance. Configuring `jbo.txn.disconnect_level` to 1 can prevent this behavior to avoid the passivation costs for this situation.

For parameters that can be configured for memory-constrained systems, see Table 8–9.

Table 8–9 AM Pool Sizing Configurations - Memory Considerations

Parameter	Description
Initial Pool Size `jbo.ampool.initpoolsize`	Set this to a low value to conserve memory at the cost of slower performance when additional AM instances are required. The default value of 0 (zero) does not create any AM instances when the AM pool is initialized.
Maximum Pool Size `jbo.ampool.maxpoolsize`	Configure this to prevent the number of AM instance from exceeding the determined value. However, if this is set too low, then some users may see an error accessing the application if no AM instances are available.
Minimum Available Pool Size `jbo.ampool.minavailablesize`	Set to 0 (zero) to shrink the pool to contain no instances when all instances have been idle for longer than the idle time out after a resource cleanup. However, a setting of 1 is commonly used to avoid the costs of re-creating the AM pool.
Maximum Available Pool Size `jbo.ampool.maxavailablesize`	Configure this to leave the maximum number of available instances specified after a resource cleanup.

For parameters that can be configured to reduce the load on the CPU to some extent through a few parameters, see Table 8–10.

Table 8–10 AM Pool Sizing Configurations - CPU Considerations

Parameter	Description
`jbo.ampool.initpoolsize`	Set this value to the number of AM instances you want the application pool to start with. Creating AM instances during initialization takes the CPU processing costs of creating AM instances during the initialization instead of on-demand when additional AM instances are required.
`jbo.recyclethreshold`	Configure this value to maintain the AM instance's affinity to a user's session. Maintaining this affinity as much as possible save the CPU processing cost of needing to switch an AM instance from one user session to another.

8.3.1.6.2 Configuring AM Pool Sizing The Application Module pool sizing configuration is largely dependant on the number of concurrent users you expect to have. To prevent performance issues, you need to make sure AM pool size is sufficient to serve all concurrent users.

Caution: The following example assumes at least 100 concurrent users. Always consult your own use case scenarios to determine the appropriate settings for your deployment.

To configure these parameters, open the `setDomainEnv.sh` file for the WebLogic Server instance and find these lines:

```
JAVA_OPTIONS="${JAVA_OPTIONS}"
export JAVA_OPTIONS
```

Replace these lines with the following:

```
JAVA_OPTIONS="-Djbo.ampool.doampooling=true
-Djbo.ampool.minavailablesize=1
-Djbo.ampool.maxavailablesize=120
-Djbo.recyclethreshold=60
-Djbo.ampool.timetolive=-1
-Djbo.load.components.lazily=true
-Djbo.doconnectionpooling=true
-Djbo.txn.disconnect_level=1
```

```
-Djbo.connectfailover=false
-Djbo.max.cursors=5
-Doracle.jdbc.implicitStatementCacheSize=5
-Doracle.jdbc.maxCachedBufferSize=19 ${JAVA_OPTIONS}"
```

Note: To limit performance implications, set the `ampool.maxavailablesize` to a value that is at least 20% more than the maximum number of concurrent users you expect in your own use case scenarios.

8.3.1.6.3 AM Pool Resource Cleanup Configurations These parameters affect the frequency and characteristics for AM pool resource cleanups.

For memory-constrained systems, configure the AM pool to clean up more AM instances more frequently so that the memory consumed by the AM instance can be freed for other purposes. However, reducing the number of available AM instances and increasing the frequency of cleanups can result in higher CPU usage and longer response times. See Table 8–11 for more information.

Table 8–11 AM Pool Resource Cleanup Configurations - Memory Considerations

Parameter	Description
`jbo.ampool.minavailablesize`	A setting of 0 (zero) shrinks the pool to contain no instances when all instances have been idle for longer than the idle time out. However, a setting of 1 is commonly used to avoid the costs of re-creating the AM pool
`jbo.ampool.maxavailablesize`	A lower value generally results in more AM instances being removed from the pool on a cleanup.
`jbo.ampool.timetolive`	A lower value reduces the time an AM instance can exist before it must be removed at the next resource cleanup.
`jbo.ampool.maxinactiveage`	A low value results in more AM instances being marked as a candidate for removal at the next resource cleanup.
`jbo.ampool.monitorsleepinterval`	This controls how frequent resource cleanups can be triggered. Configuring a lower interval results in inactive AM instances being removed more frequently to save memory.

The AM pool can be configured to reduce the need for CPU processing by allowing more AM instances to exist in the pool for longer periods of time. This generally comes at the cost of consuming more memory.

Table 8–12 AM Pool Resource Cleanup Configurations - CPU Considerations

Parameter	Description
`jbo.ampool.minavailablesize` and `jbo.ampool.maxavailablesize`	Setting these to a higher value leaves more idle instances in the pool, so that AM instances do not have to be recreated at a later time. However, the values should not be set excessively high to keep more AM instances than can be required at maximum load.
`jbo.ampool.timetolive`	A higher value increases the time an AM instance can exist before it must be removed at the next resource cleanup.
`jbo.ampool.maxinactiveage`	A higher value results in fewer AM instances being marked as a candidate for removal at the next resource cleanup.
`jbo.ampool.monitorsleepinterval`	Configuring a higher interval results in less frequent resource cleanups.

8.3.1.7 ADFc: Region Usage

Adding regions to a page can be a powerful addition to the application. However, regions can be a resource-intensive component on the page. For better performance, consider using regions only when the specific functionality is required.

8.3.1.8 Defer Task Flow Execution

By default, task flows are activated when the page is loaded, even when the task flow is not initially rendered. This causes unnecessary overhead if the task flow is never displayed.

> **Note:** For regions and taskflows, the amount of time it takes to evaluate the current viewId and the time it takes to calculate input parameters to the flow can impact your overall performance. Consider this during your design phase.

8.3.1.9 Task Flow in a Popup

By default, the child components under a popup are created even when popup is not accessed. To avoid this overhead, consider the following:

- Set childCreation to deferred

  Set childCreation="deferred" on the popup

  Set activation="deferred" on the taskflow

> **Caution:** This approach cannot be used if any of the following tags are present inside the popup:
>
> - f:attribute
> - af:setPropertyListener
> - af:clientListener
> - af:serverListener
>
> t also cannot be used if you need to refer to any child components of the popup before the popup is displayed. Setting childCreation="deferred" will postpone creating any child components of the popup and you cannot refer to them until after the popup is shown. In that case, use Conditional Activation as described below:

- Use Conditional Activation

  Add property listener on the popup in the jsff to set a condition

  Set activation="conditional" on the taskflow

  Set activate=<condition> on the taskflow

8.3.1.10 Configuring the Task Flow Inside Switcher

By default, task flows under switchers are activated when the page is loaded, not when the switcher facet is displayed. To avoid this, use conditional activation and set "active" to an expression language (EL) expression that returns 'true' when the facet is displayed.

8.3.1.11 Reusing Static Data

If the application contains static data that can be reused across the application, the cache data can be collected using a shared application module. More information on creating and using shared application modules can be found in "Sharing Application Module View Instances" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

8.3.1.12 Conditional Validations

For resource-intensive validations on entity attributes, consider using preconditions to selectively apply the validations only when needed. The cost of validation must be weighted against the cost of the precondition to determine if the precondition is beneficial to the performance. More information on specifying preconditions for validation can be found in "How to Set Preconditions for Validation" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

9 Oracle TopLink (EclipseLink) JPA Performance Tuning

This chapter describes some of the available performance tuning features for EclipseLink, an open-source persistence framework used with Oracle TopLink. The chapter includes the following topics:

Note: For more information on performance tuning in these areas, see the following:

- EclipseLink Performance Tuning at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Advanced_JPA_Development/Performance`
- Performance Monitoring and Profiling at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Advanced_JPA_Development/Performance/Performance_Profiling`
- Introduction to Optimization at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Advanced_JPA_Development/Performance#Identifying_General_Performance_Optimization`

9.1 About Oracle TopLink and EclipseLink

Oracle TopLink includes the open source EclipseLink as the Java Persistence API (JPA) implementation. Oracle TopLink extends EclipseLink with advanced integration into the Oracle Application Server.

The Java Persistence API is a specification for persistence in Java EE and Java SE applications. In JPA, a persistent class is referred to as an entity. An entity is a plain old Java object (POJO) class that is mapped to the database and configured for usage through JPA using annotations, persistence XML, or both. This chapter focuses on tuning JPA in the context of EJB3.0 and a Java EE environment.

The information in this chapter assumes that you are familiar with the basic functionality of EclipseLink. Before you begin tuning, consider reviewing the following introductory information:

- The EclipseLink JPA User's Guide at
`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA`
- "Considering JPA Entity Architecture" at
`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Introductio n/Architecture`
- Introduction to EclipseLink Queries at
`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_ Development/Querying`
- Introduction to Cache at
`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_ Development/Caching`
- Introduction to Mapping and Configuration at
`http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_ Development/Mapping`

For more information on Oracle TopLink, see the TopLink page on OTN
`http://www.oracle.com/technology/products/ias/toplink/index.html`
.

[Note that as of Oracle TopLink Release 11*g*, the older Toplink APIs have been deprecated. For more information, see the TopLink Release Notes at
`http://www.oracle.com/technology/products/ias/toplink/doc/11110/ relnotes/toplink-relnotes.html#CHDGAEDJ`]

Note: This chapter serves as a 'Quick Start' guide to performance tuning JPA in the context of a Java EE environment. While the chapter provides common performance tuning considerations and related documentation resources, it is not meant to be comprehensive list of areas to tune.

9.2 Basic Tuning Considerations

The following tuning recommendations are applicable to most deployments. Always consult your own usecase scenarios before implementing any of these configurations.

- Creating Efficient SQL Statements and Queries
- Tuning Cache Configuration
- Tuning the Mapping and Descriptor Configurations
- Using Data Partitioning

9.2.1 Creating Efficient SQL Statements and Queries

This section covers using efficient SQL statements and SQL querying. Table 9–1 and Table 9–2 show tuning parameters and performance recommendations related to SQL statements and querying.

Table 9–1 EJB/JPA Using Efficient SQL Statements and Querying

Tuning Parameter	Description	Performance Notes
Parameterized SQL Binding	Using parameterized SQL and prepared statement caching, you can improve performance by reducing the number of times the database SQL engine parses and prepares SQL for a frequently called query. EclipseLink enables parameterized SQL by default. However, not all databases and JDBC drivers support these options. Note that the Oracle JDBC driver bundled with Oracle Application Server does support this option. The persistence property in persistence.xml "eclipselink.jdbc.bind-parameters" is used to configure this. See Also: "Caching" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Caching` and "Querying" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying` Default Value: PERSISTENCE_UNIT_DEFAULT (which is true by default)	Leave parameterized SQL binding enabled for selected databases and JDBC drivers that support these options.
JDBC Statement Caching	Statement caching is used to lower the performance impact of repeated cursor creation and repeated statement parsing and creation; this can improve performance for applications using a database. Note: For Java EE applications, use the data source's statement caching (and do not use EclipseLink Statement Caching for EJB3.0/JPA, for example: `eclipselink.jdbc.cache-statements"="true"`). Set this option in an Oracle Weblogic data-source by setting `Statement Cached Type` and `Statement Cached Size` configuration options. See also "Increasing Performance with the Statement Cache" in *Administering JDBC Data Sources for Oracle WebLogic Server*. Default Value: The Oracle Weblogic Server data source default statement cache size is 10 statements per connection.	You should always enable statement caching if your JDBC driver supports this option. The Oracle JDBC driver supports this option.

Table 9–1 (Cont.) EJB/JPA Using Efficient SQL Statements and Querying

Tuning Parameter	Description	Performance Notes
Fetch Size	The JDBC fetch size gives the JDBC driver a hint as to the number of rows that should be fetched from the database when more rows are needed. For large queries that return a large number of objects, you can configure the row fetch size used in the query to improve performance by reducing the number database hits required to satisfy the selection criteria. Most JDBC drivers use a default fetch size of 10. If you are reading 1000 objects, increasing the fetch size to 256 can significantly reduce the time required to fetch the query's results. Note: The default value means use the JDBC driver default value, which is typically 10 rows for the Oracle JDBC driver. To configure this, use query hint `"eclipselink.jdbc.fetch-size"`. Default Value: 0	The optimal fetch size is not always obvious. Usually, a fetch size of one half or one quarter of the total expected result size is optimal. Note that if you are unsure of the result set size, incorrectly setting a fetch size too large or too small can decrease performance.
Batch Writing	Batch writing can improve database performance by sending groups of `INSERT`, `UPDATE`, and `DELETE` statements to the database in a single transaction, rather than individually. The persistence property in persistence.xml `"eclipselink.jdbc.batch-writing"="JDBC"` is used to configure this. Default Value: Off	Enable for the persistence unit.
Change Tracking	This is an optimization feature that lets you tune the way EclipseLink detects changes in an Entity. Default Value: AttributeLevel if using weaving (Java EE default), otherwise Deferred.	Leave at default AttributeLevel for best performance.
Weaving	Can disable through persistence.xml properties `"eclipselink.weaving"` Default Value: On	Leave on for best performance.

Table 9–1 (Cont.) EJB/JPA Using Efficient SQL Statements and Querying

Tuning Parameter	Description	Performance Notes
Read Only	Setting an EJB3.0 JPA Entity to read-only ensures that the entity cannot be modified and enables EclipseLink to optimize unit of work performance. Set through query hint "eclipselink.read-only". Can also be set at entity level using `@ReadOnly` class annotation. Default Value: False	For optimal performance use read-only on any query where the resulting objects are not changed.
firstResult and maxRows	These are JPA query properties that are used for paging large queries. Typically, these properties can be used when the entire result set of a query returning a large number of rows is not needed. For example, when a user scans the result set (a page at a time) looking for a particular result and then discards the rest of the data after the record is found.	Use on queries that can have a large result set and only a subset of the objects is needed.
Sequence number pre-allocation	Sequence number pre-allocation enables a batch of ids to be queried from the database simultaneously in order to avoid accessing the database for an id on every insert. Default Value: 50	Always use sequence number pre-allocation for best performance for inserts. `SEQUENCE` or `TABLE` sequencing should be used for optimal performance, not `IDENTITY` which does not allow pre-allocation.

9.2.1.1 Tuning Entity Relationships Query Parameters

Table 9–2 shows the Entity relationship query parameters for performance tuning.

Table 9–2 EJB3.0 Entity Relationship Query Performance Options

Tuning Parameter	Description	Performance Notes
Batch Fetching	The eclipselink.batch hint supplies EclipseLink with batching information so subsequent queries of related objects can be optimized in batches instead of being retrieved one-by-one or in one large joined read. Batch fetching has three types: JOIN, EXISTS and IN. The type is set through the query hint "eclipselink.batch.type" Note that batching is only allowed on queries that have a single object in their select clause. The query hint to configure this is "eclipselink.batch". Batch fetching can also be set using the @BatchFetch annotation. Default Value: Off	Use for queries of tables with columns mappings to table data you need.You should only use either batch fetching or joining if you know that you are going to access all of the data; if you do not intend to access the relationships, then just let indirection defer their loading. Batch fetching is more efficient than joining because it avoids reading duplicate data; therefore for best performance for queries where batch fetching is supported, consider using batch fetching instead of join reading.
Join Fetching	Join fetching is a query optimization feature that enables a single query for a class to return the data to build the instances of that class and its related objects. Use this feature to improve query performance by reducing database access. By default, relationships are not join-read: each relationship is fetched separately when accessed if you are using lazy-loading, or as a separate database query if you are not using lazy-loading. You can specify the use of join in `JPQL (JOIN FETCH)`, or you can set it multi-level in a query hint, "eclipselink.join-fetch". It also can be set in the mapping annotation @JoinFetch. Joining is part of the JPA specification, whereas batch fetching is not. And, joining works on queries that not work with batch fetching. For example, joining works on queries with multiple objects in the select clause, queries with a single result, and for cursors and first/max results, whereas batch fetching does not. See Also: "Join Fetch" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/Query_Hints#Join_Fetch` Default Value: Not Used	Use for queries of tables with columns mappings to table data you need.You should only use either batch fetching or joining if you know that you are going to access all of the data; if you do not intend to access the relationships, then just let indirection defer their loading.For the best performance of selects, where batch fetching is not supported, a join is recommended

Table 9–2 (Cont.) EJB3.0 Entity Relationship Query Performance Options

Tuning Parameter	Description	Performance Notes
Lazy loading	Without lazy loading on, when EclipseLink retrieves a persistent object, it retrieves all of the dependent objects to which it refers. When you configure lazy reading (also known as indirection, lazy loading, or just-in-time reading) for an attribute mapped with a relationship mapping, EclipseLink uses an indirection object as a place holder for the referenced object. EclipseLink defers reading the dependent object until you access that specific attribute. This can result in a significant performance improvement, especially if the application is interested only in the contents of the retrieved object, rather than the objects to which it is related. See Also: "Lazy Loading" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Mapping/Basic_Mappings/Lazy_Basics` Default Value: On for collection mapping (ToMany mappings, @OneToMany, @ManyToMany) Default Value: Off for reference (ToOne mappings, @OneToOne, @ManyToOne) (Note that setting lazy loading On for @OneToOne, @ManyToOne requires weaving, which is On by default for Java Java EE.)	Use lazy loading for all mappings. Using lazy loading and querying the referenced objects using batch fetching or Join is more efficient than Eager loading. You may also consider using optimized loading with LoadGroups which allows a query to force instantiation of relationships.

9.2.2 Tuning Cache Configuration

This section describes tuning the default internal cache that is provided by EclipseLink. Oracle Toplink/EclipseLink can also be integrated with Oracle Coherence. For information on configuring and tuning an EclipseLink Entity Cache using Oracle Coherence, see Section 9.3.1, "Integrating with Oracle Coherence".

The default settings for EJB3.0/JPA used with the EclipseLink persistence manager and cache are no locking, no cache refresh, and cache-usage `DoNotCheckCache`. To ensure that your application uses the cache and does not read stale data from the cache (when you do not have exclusive access), you must configure these and other isolation related settings appropriately. Table 9–3 shows the cache configuration options.

For more information on cache configuration, see "Caching" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Caching`.

Note: By default, EclipseLink assumes that your application has exclusive access to the data it is using (that is, there are no external, non-EclipseLink, applications modifying the data). If your application does not have exclusive access to the data, then you must change some of the defaults from Table 9–3.

Table 9–3 EJB3.0 JPA Entities and Cache Configuration Options

Tuning Parameter	Description	Performance Notes
Object Cache	EclipseLink sessions provide an object cache. EJB3.0 JPA applications that use the EclipseLink persistence manager create EclipseLink sessions that by default use this cache. This cache, known as the **session cache**, retains information about objects that are read from or written to the database, and is a key element for improving the performance of an EclipseLink application. Typically, a server session's object cache is shared by all client sessions acquired from it. Isolated sessions provide their own session cache isolated from the shared object cache. The annotation type @Cacheable specifies whether an entity should be cached. Caching is enabled when the value of the persistence.xml caching element is ENABLE_SELECTIVE or DISABLE_SELECTIVE. The value of the Cacheable annotation is inherited by subclasses; it can be overridden by specifying Cacheable on a subclass. Cacheable(false) means that the entity and its state must not be cached by the provider. Default Value: Enabled (shared is True)	Generally it is recommended that you leave caching enabled. If you have an object that is always read from the database, as in a pessimistic locked object, then the cache for that entity should be disabled. Also, consider disabling the cache for infrequently accessed entities

Query Result Set Cache	In addition to the object cache in EclipseLink, EclipseLink also supports a query cache:
	■ The object cache indexes objects by their primary key, allowing primary key queries to obtain cache hits. By using the object cache, queries that access the data source can avoid the cost of building the objects and their relationships if the object is already present.
	■ The query cache is distinct from the object cache. The query cache is indexed by the query and the query parameters - not the object's primary key. This enables any query executed with the same parameters to obtain a query cache hit and return the same result set.
	
	The query hints for a query cache are:
	`"eclipselink.query-cache"`
	`"eclipselink.query-cache.size"`
	`"eclipselink.query-cache.invalidation"`
	
	See Also: "Caching" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Caching` and "EclipseLink JPA Query Hints" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Querying/Query_Hints`
	
	Default Value: Not Used
		Use for frequently executed non-primary key queries with infrequently changing result sets.Use with a cache invalidation time out to refresh as needed.

Table 9–3 (Cont.) EJB3.0 JPA Entities and Cache Configuration Options

Tuning Parameter	Description	Performance Notes

Cache Size	Cache size can be configured through persistence properties: `"eclipselink.cache.size.<entity>"`
	`"eclipselink.cache.size.default"`
	`"eclipselink.cache.type.default"`
	See Also: "Configuring Persistence Units Using persistence.xml" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Configuration/JPA/persistence.xml` and 'Class PersistenceUnitProperties" at `http://www.eclipse.org/eclipselink/api/2.3/org/eclipse/persistence/config/PersistenceUnitProperties.html`
	Default Value: Type `SoftWeak`, Size 100 (per Entity).
		Set the cache size relative to how much memory you have available, how many instances of the class you have, the frequency the entities are accessed, and how much caching you want based on your tolerance for stale data.
		Consider creating larger cache sizes for entities that have many instances that are frequently accessed and stale data is not a big issue.
		Consider using smaller cache sizes or no cache for frequently updated entities that must always have fresh data, or infrequently accessed entities.

Locking	Oracle supports the locking policies shown in Table 9–4: no locking, optimistic, pessimistic, and read-only. Locking is set through JPA @Version annotation, eclipselink.read-only How to Use EclipseLink Locking at `http://wiki.eclipse.org/EclipseLink/Examples/JPA/Locking` Default Value: No Locking	For entities that can be updated concurrently, consider using the locking policy to prevent a user from writing over another users changes. To optimize performance for read-only entities, consider defining the entity as read-only or use a read-only query hint.

Table 9–3 (Cont.) EJB3.0 JPA Entities and Cache Configuration Options

Tuning Parameter	Description	Performance Notes
Cache Usage	By default, all query types search the database first and then synchronize with the cache. Unless refresh has been set on the query, the cached objects can be returned without being refreshed from the database. You can specify whether a given query runs against the in-memory cache, the database, or both. To get performance gains by avoiding the database lookup for objects already in the cache, you can configure that the search attempts to retrieve the required object from the cache first, and then search the data source only if the object is not in the cache. For a query that looks for a single object based on a primary key, this is done by setting the query hint "`eclipselink.cache-usage`" to `CheckCacheByExactPrimaryKey`. Default Value: `DoNotCheckCache`	For faster performance on primary key queries, where the data is typically in the cache and does not require a lot of refreshing, it is recommended to check the cache first on these queries (using `CheckCacheByExactPrimaryKey`). This avoids the default behavior of retrieving the object from the database first and then for objects already in the cache, returning the cached values (not updated from the database access, unless refresh has been set on the query).
Isolation	There is not a single tuning parameter that sets a particular database transaction isolation level in a JPA application that uses EclipseLink. In a typical EJB3.0 JPA application, a variety of factors affect when database transaction isolation levels apply and to what extent a particular database transaction isolation can be achieved, including the following: ■ Locking mode ■ Use of the Session Cache ■ External Applications ■ Database Login method `setTransactionIsolation` See Also: "Shared and Isolated Cache" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Caching/Shared_and_Isolated`	

Table 9–3 (Cont.) EJB3.0 JPA Entities and Cache Configuration Options

Tuning Parameter	Description	Performance Notes
Cache Refreshing	By default, EclipseLink caches objects read from a data source. Subsequent queries for these objects access the cache and thus improve performance by reducing data source access and avoiding the cost of rebuilding object's and their relationships. Even if a query accesses the data source, if the objects corresponding to the records returned are in the cache, EclipseLink uses the cached objects. This default caching policy can lead to stale data in the application. Refreshing can be enabled at the entity level (`alwaysRefresh` or `refreshOnlyIfNewer` and `expiry`) and at the query level (with the `eclipselink.refresh` query hint). You can also force queries to go to the database with (`disableHits`). Using an appropriate locking policy is the only way to ensure that stale or conflicting data does not get committed to the database. For more information see: Section 9.2.2.1, "Cache Refreshing Scenarios" See Also: "Caching Overview" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Caching/Caching_Overview` Default Value: No Cache Refreshing	Try to avoid entity level cache refresh and instead, consider configuring the following: ■ cache refresh on a query-by-query basis ■ cache expiration ■ isolated caching

9.2.2.1 Cache Refreshing Scenarios

There are a few scenarios to consider for data refreshing in the cache, all with performance implications:

- In the case where you never want cached data and always want fresh data, consider using an isolated cache (Shared=False). This is the case when certain data in the application changes so frequently that it is desirable to always refresh the data, instead of only refreshing the data when a conflict is detected.
- In the case when you want to avoid stale data, but getting stale data is not a major issue, then using a cache expiry policy would be the recommended solution. In this case you should also use optimistic locking, which automatically refresh stale objects when a locking error occurs. If using optimistic locking, you could also enable the entity @Cache attributes `alwaysRefresh` and `refreshOnlyIfNewer` to allow queries that access the database to refresh any stale objects returned, and avoid refreshing invalid objects when unchanged. You may also want to enable refreshing on certain query operations when you know you want refreshed data, or even provide the option of refreshing something from the client that would call a refreshing query.
- In the case when you are not concerned about stale data, you should use optimistic locking; this automatically refresh stale objects in the cache on locking errors.

9.2.2.2 Tuning the Locking Mode Policies

The locking modes, as shown in Table 9–4, along with EclipseLink cache-usage and query refreshing options, ensures data consistency for EJB entities using JPA. The different combinations have both functional and performance implications, but often the functional requirements for up-to-date data and data consistency lead to the settings for these options, even when it may be at the expense of performance.

For more information, see "Locking" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Mapping/Locking`.

Table 9–4 Locking Mode Policies

Locking Option	Description	Performance Notes
No Locking	The application does not prevent users overwriting each other's changes. This is the default locking mode. Use this mode if the Entity is never updated concurrently or concurrent reads and updates to the same rows with read-committed semantics is sufficient. Default Value: No Locking	In general, no locking is faster, but may not meet your needs for data consistency.
Optimistic	All users have read access to the data. When a user attempts to make a change, the application checks to ensure the data has not changed since the user read the data. See Also: "Optimistic Locking" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Mapping/Locking/Optimistic_Locking`	If infrequent concurrent updates to the same rows are expected, then optimistic locking may provide the best performance while providing data consistency guarantees.
Pessimistic	The first user who accesses the data with the purpose of updating it locks the data until completing the update.	If frequent concurrent updates to the same rows are expected, pessimistic locking may be faster than optimistic locking that is getting a lot of concurrent access exceptions and retries. When using pessimistic locking at the entity level, it is recommended that you use it with an isolated cache (Shared=False) for best performance.
Read Only	Setting an EJB3.0 JPA Entity to read-only ensures that the entity cannot be modified and enables EclipseLink to optimize unit of work performance. Set at the entity level using @ReadOnly class annotation. Can also be set at the query level through query hint "`eclipselink.read-only`".	Defining an entity as read-only can perform better than an entity that is not defined as read-only, yet does no inserts, updates, or deletes, since it enables EclipseLink to optimize the unit of work performance. Always use read-only for all read-only operations

9.2.3 Tuning the Mapping and Descriptor Configurations

EclipseLink can transform data between an object representation and a representation specific to a data source. This transformation is called mapping and it is the core of a EclipseLink project.

A mapping corresponds to a single data member of a domain object. It associates the object data member with its data source representation and defines the means of performing the two-way conversion between object and data source.

For information on Mapping see, "Configuring Mappings" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Basic_JPA_Development/Mapping`.

9.2.4 Using Data Partitioning

EclipseLink allows you to configure data partioning using the @Partitioned annotation. Partitioning enables an application to scale information across multiple databases; including clustered databases. For more information on using @Partioned and other partitioning policy annotations, see "Data Partitioning" at `http://wiki.eclipse.org/EclipseLink/UserGuide/JPA/Advanced_JPA_Development/Data_Partitioning`.

9.3 Advanced Tuning Considerations

After you have performed the modifications recommended in the previous section, you can make additional changes that are specific to your deployment. Consider carefully whether the recommendations in this section are appropriate for your environment.

- Integrating with Oracle Coherence
- Analyzing EclipseLink JPA Entity Performance

9.3.1 Integrating with Oracle Coherence

Oracle Toplink can be integrated with Oracle Coherence. This integration is provided through the Oracle TopLink Grid feature. With TopLink Grid, there are several types of integration with EclipseLink JPA features.

For example:

- Replace the default EclipseLink L2 cache with Coherence. This provides support for very large L2 caches that span cluster nodes. EclipseLink's default L2 cache improves performance for multi-threaded and Java EE server hosted applications running in a single JVM, and requires configuring special cache coordination features if used across a cluster.
- Configure entities to execute queries in the Coherence data grid instead of the database. This allows clustered application deployments to scale beyond database-bound operations.

For more information on using EclipseLink JPA with a Coherence Cache, see "JPA on the Grid" Approach at `http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/tlgug003.htm`

For more information on Oracle Toplink integration with Oracle Coherence, see "Oracle TopLink Integration with Coherence Grid Guide" at `http://www.oracle.com/technology/products/ias/toplink/doc/11110/grid/toc.htm`

9.3.2 Analyzing EclipseLink JPA Entity Performance

This section lists a few features in EclipseLink that can help you analyze your JPA application performance:

- Form monitoring performance, see "Performance Monitoring" in the `EclipseLink User's Guide`. Note that this tool is intended to profile and monitor information in a multithreaded server environment.
- For profiling performance, see "Measuring EclipseLink Performance with the EclipseLink Profiler" in the `EclipseLink User's Guide`. Note that this tool is intended for use with single-threaded finite use cases.

- For debugging performance issues and testing, you can view the SQL generated from EclipseLink. To view the SQL, increase the logging level to "FINE" by using the EclipseLink JPA extensions for logging.

  For best performance, remember to restore the logging levels to the default levels when you are done profiling or debugging.