IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

ORACLE CORPORATION and ORACLE U.S.A. INC.,	)	
Plaintiffs/Counterclaim Defendants,	)	C.A. No. 06-414-SLR
v.	)	**JURY TRIAL DEMANDED**
EPICREALM LICENSING, LP,	)	**PUBLIC VERSION**
Defendant/Counterclaim Plaintiff.	)	

APPENDIX TO EPICREALM'S BRIEF IN OPPOSITION TO ORACLE'S MOTION TO EXCLUDE EXPERT REPORT AND TESTIMONY OF EPICREALM'S DAMAGES EXPERT, MICHAEL J. WAGNER

Richard L. Horwitz (#2246)
David E. Moore (#3983)
POTTER ANDERSON & CORROON LLP
Hercules Plaza, 6th Floor
1313 N. Market Street
Wilmington, DE 19899
Tel: (302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Attorneys for Defendant/Counterclaim Plaintiff epicRealm Licensing, LP

OF COUNSEL:

Harry J. Roper
George S. Bosy
Aaron A. Barlow
Patrick L. Patras
David R. Bennett
Paul D. Margolis
Benjamin J. Bradford
Emily C. Johnson
JENNER & BLOCK
330 N. Wabash Avenue
Chicago, IL 60611-7603
Tel: (312) 923-8305

Dated: August 21, 2008
Public Version Dated: August 28, 2008
880359 / 31393 / Oracle

TABLE OF CONTENTS

Exh.	*Document*
1.	Expert Report of Michael J. Wagner, May 2, 2008
2.	Deposition transcript of Michael J. Wagner, June 25, 2008
3.	Excerpt from deposition transcript of Lakshminarayanan Chidambaran, Feb. 29, 2008
4.	Excerpt from deposition transcript of Steve Harris, April 18, 2008
5.	Excerpt from Expert Witness Rebuttal Report on Damages by Terry L. Musika and Exhibit 4, June 6, 2008
6.	Excerpt from deposition transcript of Terry L. Musika, July 1, 2008
7.	Excerpt from deposition transcript of Mark Nelson, Feb. 22, 2008
8.	Excerpt from deposition transcript of Timothy Willis, May 21, 2008
9.	Certificate of Amendment of Certificate of Incorporation, Jan 31, 2000, EPIC057149-057151
10.	License Agreement between InfoSpinner and IBM, 1/30/98, EPIC034717-034746
11.	Email from J. Anthony to M. Denning, 1/21/98, IBMOER00707
12.	Memo to file re IBM/InfoSpinner License Agreement, 3/9/98, IBMOER00710-00712
13.	Passport Advantage Numbers, IBMOER00790-00792
14.	Presentation, Oracle Application Server 10g Release 2 From Simple Clustering to Resiliency, ORCL 01050220-01050280
15.	Excerpt from deposition transcript of Hal Hildebrand, Dec. 21, 2007
16.	Excerpt from deposition transcript of David Finkel, Feb. 12, 2008
17.	Excerpt from deposition transcript of Mark Clark, April 17, 2008
18.	Letter from J. Moore, FreshDirect, 3/11/08, EPIC436363
19.	Oracle Application Server 10g, Best Practices 10g (9.0.4) B12223-01, May 2004, ORCL00384158-00384305

IN THE UNITED STATES DISTRICT COURT
FOR THE DISTRICT OF DELAWARE

CERTIFICATE OF SERVICE

I, David E. Moore, hereby certify that on August 28, 2008, the attached document was electronically filed with the Clerk of the Court using CM/ECF which will send notification to the registered attorney(s) of record that the document has been filed and is available for viewing and downloading.

I further certify that on August 28, 2008, the attached document was Electronically Mailed to the following person(s):

Mary B. Graham
Morris, Nichols, Arsht & Tunnell LLP
1201 North Market Street
P.O. Box 1347
Wilmington, DE 19899
MGraham@MNAT.com

James G. Gilliland
Igor Shoiket
Townsend and Townsend and Crew LLP
Two Embarcadero Center
Eighth Floor
San Francisco, CA 94111-3834
OracleEpicrealm@townsend.com

Theodore T. Herhold
Robert J. Artuz
Eric M. Hutchins
Eric A. Mercer
Joseph A. Greco
Nitin Gupta
Townsend and Townsend and Crew LLP
379 Lytton Avenue
Palo Alto, CA 94301
OracleEpicrealm@townsend.com

Chad E. King
Townsend and Townsend and Crew LLP
1200 Seventeenth Street
Suite 2700
Denver, CO 80202
OracleEpicrealm@townsend.com

/s/ David E. Moore
Richard L. Horwitz
David E. Moore
Potter Anderson & Corroon LLP
Hercules Plaza – Sixth Floor
1313 North Market Street
P.O. Box 951
Wilmington, DE 19899-0951
(302) 984-6000
rhorwitz@potteranderson.com
dmoore@potteranderson.com

Exhibit 1

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 2

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 3

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 4

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 5

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 6

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 7

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 8

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 9

CERTIFICATE OF AMENDMENT OF CERTIFICATE OF INCORPORATION

InfoSpinner, Inc., a corporation organized and existing under the General Corporation Law of the State of Delaware (the "Corporation"),

DOES HEREBY CERTIFY:

FIRST: That this corporation was originally incorporated on November 8, 1995, pursuant to the General Corporation Law of the State of Delaware (the "General Corporation Law").

SECOND: That the Company's Certificate of Incorporation is amended by deleting in its entirety existing ARTICLE I and inserting in lieu thereof a new ARTICLE I, reading as follows:

"ARTICLE I

The name of this corporation is EpicRealm Inc."

THIRD: The foregoing amendment was approved by the holders of the requisite number of shares of the Corporation in accordance with Section 228 of the General Corporation Law and written notice of such approval of this Certificate of Amendment of the Company's Certificate of Incorporation has been given in accordance with the provisions of Section 228 of the General Corporation Law to those holders of the Corporation's stock who have not so approved this Certificate of Amendment.

* * * * *

d-736589.1

02/03/00 THU 18:05 [TX/RX NO 8424]
02/04/00 FRI 15:20 [TX/RX NO 8498]

IN WITNESS WHEREOF, this Certificate of Amendment of Certificate of Incorporation has been signed by the President of this corporation this 27th day of January, 2000.

INFOSPINNER, INC.
(to be renamed hereby
EPICREALM INC.)

By: _____
John D. Ferguson
President and CEO

State of Delaware

Office of the Secretary of State

I, EDWARD J. FREEL, SECRETARY OF STATE OF THE STATE OF DELAWARE, DO HEREBY CERTIFY THE ATTACHED IS A TRUE AND CORRECT COPY OF THE CERTIFICATE OF AMENDMENT OF "INFOSPINNER, INC.", CHANGING ITS NAME FROM "INFOSPINNER, INC." TO "EPICREALM INC.", FILED IN THIS OFFICE ON THE FOURTH DAY OF FEBRUARY, A.D. 2000, AT 1 O'CLOCK P.M.

A FILED COPY OF THIS CERTIFICATE HAS BEEN FORWARDED TO THE NEW CASTLE COUNTY RECORDER OF DEEDS.

Edward J. Freel, Secretary of State

2558538 8100

001058164

AUTHENTICATION: 0239571

DATE: 02-04-00

02/04/00 FRI 16:20 [TX/RX NO 6498]

Exhibit 10

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 11

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 12

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 13

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 14

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 15

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 16

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 17

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 18

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 19

Oracle® Application Server 10*g*

Best Practices

10*g* (9.0.4)

Part No. B12223-01

May 2004

ORACLE

Oracle Application Server 10g Best Practices, 10g (9.0.4)

Part No. B12223-01

Copyright © 2004 Oracle. All rights reserved.

The Programs (which include both the software and documentation) contain proprietary information; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs, except to the extent required to obtain interoperability with other independently created software or as specified by law, is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. This document is not warranted to be error-free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose.

If the Programs are delivered to the United States Government or anyone licensing or using the Programs on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the Programs, including documentation and technical data, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement, and, to the extent applicable, the additional rights set forth in FAR 52.227-19, Commercial Computer Software--Restricted Rights (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and we disclaim liability for any damages caused by such use of the Programs.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

The Programs may provide links to Web sites and access to content, products, and services from third parties. Oracle is not responsible for the availability of, or any content provided on, third-party Web sites. You bear all risks associated with the use of such content. If you choose to purchase any products or services from a third party, the relationship is directly between you and the third party. Oracle is not responsible for: (a) the quality of third-party products or services; or (b) fulfilling any of the terms of the agreement with the third party, including delivery of products or services and warranty obligations related to purchased products or services. Oracle is not responsible for any loss or damage of any sort that you may incur from dealing with any third party.

4

High Availability

This chapter describes high availability best practices for Oracle Application Server. It includes the following topics:

- Section 4.1, "Distribute Identity Management Components"
- Section 4.2, "Use OPMN for Crash Handling and Monitoring"
- Section 4.3, "Analyze High Availability using iHAT or Topology Viewer"
- Section 4.4, "Use Metric Based Load Balancing to Tune High Availability"

4.1 Distribute Identity Management Components

With Oracle Application Server 10g, Oracle Identity Management can be deployed in a variety of high availability configurations such as Oracle Application Server Cold Failover Clusters (OracleAS Cold Failover Cluster) and Active Failover Cluster (AFC).

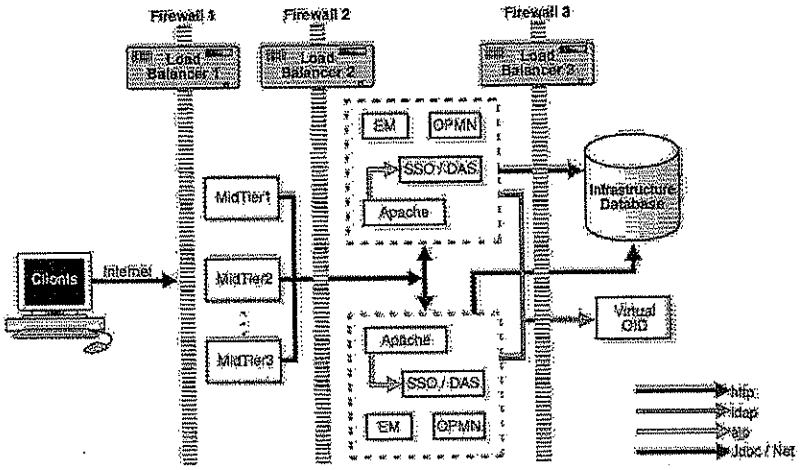
AFC refers to the deployment of Oracle Identity Management and the OracleAS Metadata Repository on hardware clusters where each node of the cluster runs the database instances as well as the ID processes and all of them access a shared RAC database as their repository.

OracleAS Cold Failover Cluster solution for the infrastructure uses a two node hardware cluster (one active node and one passive node) that in case of failure of one of the nodes, relocates the execution of an application from a failed node to a designated standby node.

For both type of configurations, the "out-of-the-box" installation for Oracle Application Server places the OracleAS Single Sign-On service, Delegated Administration Service (DAS) and DIP in the same tier as the infrastructure database itself.

Both for scalability and security reasons, Oracle recommends separating the OracleAS Single Sign-On, DAS, and DIP services from the node or nodes where the infrastructure database resides as shown in Figure 4–1. Since users access most enterprise applications from both within the company intranet as well as Internet, their access needs to be authenticated by ID in both cases. A distributed Oracle Identity Management deployment where the OracleAS Single Sign-On and DAS components reside in the DMZ meets this requirement. It is also important that while users are allowed to authenticate and access applications, their authentication data is well protected from unauthorized access from the Internet. Locating Oracle Internet Directory and the database repository for the identity data in the intranet ensures this. If necessary, the OracleAS Single Sign-On and DAS tier servers can be co-located with the middle tier servers (in separate Oracle homes) in the DMZ allowing them to be scaled in tandem with middle-tier growth.

Figure 4–1 Distributed Identity Management



In the specific case of AFC, the infrastructure database is a RAC database and a group of Oracle Internet Directorys exposes a common virtual IP by using a Hardware Load Balancer where Firewall 3 is placed.

In the case of CFC, the cluster ware running on the active and standby nodes where the infrastructure database resides expose the same virtual IP to OracleAS Single Sign-On, DAS, and middle tiers.

4.2 Use OPMN for Crash Handling and Monitoring

Oracle Application Server 10*g* allows you to configure Oracle Process Manager and Notification server (OPMN) to execute your own custom event scripts whenever a particular component starts, stops, or crashes. The code associated with crash events should:

- release any resources used by a component that has crashed
- log special characteristics associated with each crash event
- take possible actions that enhance the resiliency of the system

If the component being monitored by OPMN uses any special resources not controlled by Oracle Application Server, the crash event script should release them. A good example is any type of code in an OC4J container that uses JNI invocations. In this type of cases, OC4J does not control the resources under the JNI invocation, so that those resources may become trapped in the case of a Java Virtual Machine (JVM) crash. Common resources to control include sockets in use, memory in the JNI code, connections to database, and locked files.

The same script may take actions about the application that generated the crash. If it is likely that the crash was due to the overloading of the specific component, it could be useful to start a new instance of such a component in another node. For instance, the event script could be used for starting a new OC4J container in another application server instance that is mounting the same URL on Oracle HTTP Server. Or it could be used to relocate the repository in a file-based configuration when the DCM daemon of that specific instance crashes.

You can use OPMN to monitor your custom external process in a consistent mode with the default stack controlled by OPMN. Use OPMN to start, stop and monitor any

additional processes that are used by your system. For example, if your application uses a external resource such as an HTTP Server different from Oracle HTTP Server, you can add it to the OPMN control list so that your system High Availability is monitored in a consistent way.

Refer to the *Oracle Process Manager and Notification Server Administrator's Guide* for more information.

4.3 Analyze High Availability using iHAT or Topology Viewer

Oracle Application Server Hi-Av Tool (iHAT) and Oracle Enterprise Manager Topology Viewer are real-time, graphical monitors based on topology information maintained by all the OPMN servers running in an Oracle Application Server Farm.

iHAT is a standalone java application that runs from a local display computer. iHAT connects to a single OPMN server to monitor and control an entire Oracle Application Server Farm.

The Topology Viewer is a J2EE application that can be deployed on Oracle Application Server Control to provide a real-time view of the application server processes managed by OPMN. You can view all the instances in a Farm, including clusters and their members.

Both iHat and the Topology Viewer display performance metrics for each component.

It is a good practice to verify the system High Availability prior to the publishing of its functionality to the end-user (pre-production testing). You can leverage either iHAT or Topology Viewer capabilities, to verify:

- what is installed and configured to be managed by OPMN across an entire Oracle Application Server Farm
- what is the current status of all the processes which are part of the Oracle Application Server Farm

By using various load simulators you can stress your High Availability topology and verify from the graphical tools that any component in the stack can be started, stopped, or disabled without any impact in the system response. You should verify that the number of requests sent to the system are all completed correctly even when one of the subcomponents is stopped. This verification is very simple from iHat or Topology Viewer. They both allow you to start, stop, or disable any process controlled by OPMN and they both show the number of completed requests. You should verify that the response to all requests are correct (this should be provided by your load simulator) and that the total number of requests is being completed. This information is available on iHat and Topology Viewer.

You can learn how to use Topology Viewer and iHat at the Oracle Application Server Utilities area on OTN (`http://otn.oracle.com`).

4.4 Use Metric Based Load Balancing to Tune High Availability

mod_oc4j is the module that routes requests from the Oracle HTTP Server to OC4J through the AJP 1.3 protocol. mod_oc4j provides the load balancing mechanism between Oracle HTTP Server and the OC4J containers that share a common OC4J mount point or are accessed through the same base URL. Thus, the high availability of a configuration where several OC4J nodes are being load balanced behind an Oracle HTTP Server, is totally dependent on the mod_oc4j configuration. A load balancing policy that mod_oc4j supports in Oracle Application Server 10g is the metric based policy. When this type of policy is specified in the mod_oc4j.conf file, mod_oc4j routes

requests as per the run time metrics from OC4J processes that indicate how much load is currently being placed on the OC4J process. For this to happen, each one of the OC4J nodes continuously sends metric values to mod_oc4j. mod_oc4j then designates the node for routing of the HTTP request. However, this policy incurs a penalty on performance since the OC4J nodes are obtaining and sending metrics on a re al time basis.

If you want to maximize the usage of resources but without penalizing the performance of your system, you can use the metric based load balancing to tune your routing weights and then change the load balancing to a weighted round-robin protocol. Set up your load balancing protocol to metric-based and place the system in production or, if possible, in production simulation. Based on the number of the requests delivered to each node on a usage cycle you can make a good estimation of the weights that you should assign to each one of the nodes. Obviously, this tuning is more accurate when the number of requests and the load in each node due to external influences to the application being load balanced, do not change much during the usage cycle. You can then optimize the usage of each node with a better performance.

For more information refer to the *Oracle HTTP Server Administrator's Guide*.

5

Performance and Scalability

This chapter describes performance and scalability best practices. The topics include:

- Section 5.1, "OracleAS Web Cache Best Practices"

5.1 OracleAS Web Cache Best Practices

This chapter describes Oracle Application Server Web Cache best practices. The topics include:

- Section 5.1.1, "Improve Performance, Scalability, and Availability"
- Section 5.1.2, "Planning and Deployment"
- Section 5.1.3, "OracleAS Web Cache Security"
- Section 5.1.4, "Configuring OracleAS Web Cache"
- Section 5.1.5, "Increasing Cache Hits"
- Section 5.1.6, "Invalidation and Expiration"
- Section 5.1.7, "Optimizing Response Times"

5.1.1 Improve Performance, Scalability, and Availability

OracleAS Web Cache improves the scalability, performance and availability of e-business Web sites. Using OracleAS Web Cache, your applications benefit from higher throughput, shorter response times and lower infrastructure costs.

- Unlike legacy cache servers that only handle static data, OracleAS Web Cache combines caching, compression and assembly technologies to accelerate the delivery of both static and dynamically generated Web content.
- OracleAS Web Cache provides support for partial-page caching with Edge-Side Includes (ESI), personalization, and dynamic content assembly at the network edge See Section 5.1.5.4 for more information.
- OracleAS Web Cache includes patent-pending clustering functionality that increases capacity for content storage and ensures scalability and availability for cacheable content, even when a member cache experiences a failure or is taken offline for maintenance. See Section 5.1.2.2 for more information.
- OracleAS Web Cache also provides back-end origin server load balancing, failover, and surge protection features that ensure consistent application performance and greater overall reliability.
- OracleAS Web Cache is designed to run with commodity hardware, reducing the cost.

Using OracleAS Web Cache and its ESI features, your business application performance can improve by several orders of magnitude with very little development effort. The return on investment is also significant, both in terms of developer resources (you no longer need to build your own dynamic caching solution) and hardware cost savings.

5.1.2 Planning and Deployment

The following sections describe best practices related to planning for and deploying OracleAS Web Cache:

- Section 5.1.2.1, "Use Two CPUs and Consider Deploying on Dedicated Hardware"
- Section 5.1.2.2, "Cluster Cache Instances for Better Availability, Scalability, and Performance"
- Section 5.1.2.3, "Use a Network Load Balancer in Front of OracleAS Web Cache"
- Section 5.1.2.4, "Use OracleAS Web Cache Built-In Load Balancing for Availability and Scalability of Origin Servers"
- Section 5.1.2.5, "Deploy Caches in Remote Offices for Faster Response Times and Reduced WAN Traffic"
- Section 5.1.2.6, "Use the Latest Version"
- Section 5.1.2.7, "Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly"

5.1.2.1 Use Two CPUs and Consider Deploying on Dedicated Hardware

You can deploy OracleAS Web Cache on the same node as the application Web server or on a separate node. When making your decision, consider system resources, such as the number of CPUs. OracleAS Web Cache is designed to use one or two CPUs. Because OracleAS Web Cache is an in-memory cache, it is rarely limited by CPU cycles. Additional CPUs do not increase performance significantly. However, the speed of the processors is critical; use the fastest CPUs you can afford.

If other resources are competing with OracleAS Web Cache for CPU usage, then you should take the requirements of those resources into account when determining the number of CPUs needed. You can derive a significant performance benefit from OracleAS Web Cache running on the same node as the application Web server, although a separate node for OracleAS Web Cache is often optimal.

For a Web site with more than one OracleAS Web Cache instance, consider installing each instance on a separate two-CPU node, either as part of a cache cluster or as standalone instances. When OracleAS Web Cache instances are on separate nodes, you are less likely to encounter operating system limitations, particularly in network throughput. For example, two caches on two separate two-CPU nodes are less likely to encounter operating system limitations than two caches on one four-CPU node.

5.1.2.2 Cluster Cache Instances for Better Availability, Scalability, and Performance

To increase the availability, scalability, and performance of your Web site, you can configure multiple instances of OracleAS Web Cache to run as members of a cache cluster. A cache cluster is a loosely coupled collection of cooperating OracleAS Web Cache instances working together to provide a single logical cache.

Cache clusters provide failure detection and failover of caches, increasing the availability of your Web site. If a cache fails, other members of the cache cluster detect the failure and take over ownership of the cached content of the failed cluster member.

By distributing the Web site's content across multiple OracleAS Web Cache instances, more content can be cached and more client connections can be supported, expanding the overall capacity of your Web site and improving its performance.

For more information about cache clusters, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.2.3 Use a Network Load Balancer in Front of OracleAS Web Cache

Many customers deploy a single instance of OracleAS Web Cache in front of their application Web server farm. In such deployments, the OracleAS Web Cache acts as the virtual IP address for the application, in addition to providing caching and load balancing services. This deployment is both functionally sufficient and cost-effective for customers that do not require 100 percent application uptime. The OracleAS Web Cache is highly stable and, in the event of a failure, a process monitor will automatically restart the cache.

For customers who cannot tolerate a single point of failure, Oracle Corporation recommends that two or more nodes running OracleAS Web Cache be deployed behind a third-party hardware load balancing device. In turn, customers should use the built-in load balancing functionality in OracleAS Web Cache to distribute cache miss traffic over the application Web server farm, as described in Section 5.1.2.4. Refer to the *Oracle Application Server Web Cache Administrator's Guide* for more information about using network load balancers.

5.1.2.4 Use OracleAS Web Cache Built-In Load Balancing for Availability and Scalability of Origin Servers

Situated between Web browser clients and the origin servers, OracleAS Web Cache includes built-in weighted load balancing and failover detection features to ensure that cache misses are directed to the most available, highest performing origin server in the application Web server farm. The cache supports both stateless and stateful load balancing mechanisms, including the use of cookies and URL parameters to maintain server affinity when required. (OracleAS Web Cache can be configured to generate its own session-binding cookie, allowing you to use sessions without having to modify your applications.)

In addition, OracleAS Web Cache maintains a pool of HTTP connections between the cache and the origin Web servers to reduce connection establishment overhead and improve cache miss performance.

To avoid a single point of failure, two or more nodes running OracleAS Web Cache should be deployed behind a third-party hardware load-balancing device. However, Oracle Corporation also recommends that customers use the built-in load balancing and failure detection functionality in OracleAS Web Cache to route cache miss requests to origin servers. Deploying additional load balancing hardware between the OracleAS Web Cache and origin server tiers is not recommended for the following reasons:

- Cost: Using another tier of load balancing hardware adds significant cost to a deployment, in part because these devices must also be deployed in pairs for high availability reasons.
- Complexity: Another tier of load balancing hardware is another set of systems to configure, manage, and troubleshoot.
- Features: OracleAS Web Cache includes patent-pending performance assurance and surge protection features that enable customers to sustain higher loads with

less application and database server hardware. These features depend on the capacity-based load balancing algorithms in OracleAS Web Cache.

For more information on load balancing, performance assurance and surge protection functionality, see the *Oracle Application Server Web Cache Administrator's Guide* and various technical white papers on OracleAS Web Cache available on OTN (`http://otn.oracle.com`).

5.1.2.5 Deploy Caches in Remote Offices for Faster Response Times and Reduced WAN Traffic

OracleAS Web Cache offers hierarchical caching features that enable customers to easily create Content Delivery Networks (CDNs). For high availability and performance, many Internet businesses mirror their Web sites in strategic geographical locations. Caching is an excellent low-cost alternative to full-scale mirroring. Caching may also be used to serve local markets in order to shorten response times to these markets, and to reduce bandwidth and rack space costs for the content provider. Within the corporate intranet, so-called "Enterprise" CDNs (eCDNs) provide shorter response times for branch office users of e-business applications. Compared to application mirroring and database replication, eCDN is a more manageable and cost-effective model of distributed computing. Using OracleAS Web Cache, customers can distribute the content assembly and delivery functions of their applications to key network access points, while maintaining centralized management of application logic and data.

In setting up an eCDN, customers typically deploy OracleAS Web Cache in each branch office data center as well as in the central office where the application and database are maintained. For example, a U.S.-based company might deploy instances of OracleAS Web Cache in its U.S., Japanese, and Australian offices. The central cache residing in the U.S. serves as the origin server for the caches in Japan and Australia. Using various commercially available DNS routing techniques, requests are handled by the cache that is closest to the end user. A Web browser request made by a Japanese employee, for instance, is handled by the cache instance in Japan, thereby reducing WAN traffic and eliminating long-haul network latencies. In a distributed cache hierarchy, the central cache is aware of the branch office caches. As a result, any content invalidation messages sent to the central cache automatically propagate to the remote caches. This invalidation propagation feature ensures content consistency across the CDN and simplifies the management of cache hierarchies.

5.1.2.6 Use the Latest Version

Make sure that you are using the latest version of OracleAS Web Cache, including any available patches. Use the Upgrade tool to upgrade the old configuration to the new configuration. To check for the latest patches, go to Oracle *MetaLink*.

5.1.2.7 Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly

Although there is a growing trend to specifying the caching rules dynamically with the Surrogate-Control response header, some sites continue to use OracleAS Web Cache Manager for configuring the rules statically. Typically, this configuration is done at the start of the deployment cycle. After adequate testing in a staging area to validate the rules, OracleAS Web Cache is deployed in a production environment. Problems may arise when the backend application is upgraded for patches or with new versions and some or all of the earlier statically configured rules become not applicable and void. For example, if a site uses a session-related caching rule and, after applying a patch, the name of the session cookie or session-embedded URL parameter changes, all the

pages related to that rule will no longer be cacheable, resulting in poor performance for the site.

When applying application upgrades and patches, it is important to understand the extent of the application changes and then verify and tune the related caching rules in OracleAS Web Cache. By periodically checking the cache-hit percentage and ensuring that it remains more or less constant, you can guard against unexpected behavior. Whenever there is a major change in the database or the mid-tier layer, such as for upgrades or application patches, you should validate caching rules much the same way as you did during the initial deployment cycle, including, but not limited, to using debug-level event logging. And if possible, include OracleAS Web Cache in your application regression test cycle.

5.1.3 OracleAS Web Cache Security

The following sections describe best practices related to security issues:

- Section 5.1.3.1, "Route All HTTP and HTTPS Traffic Through OracleAS Web Cache"
- Section 5.1.3.2, "Secure Administration, Invalidation, and Statistics Monitoring Using HTTPS"
- Section 5.1.3.3, "Use Web Caching to Help Defend Against Denial-of-Service Attacks"
- Section 5.1.3.4, "Change Passwords Frequently"

5.1.3.1 Route All HTTP and HTTPS Traffic Through OracleAS Web Cache

In general, you should route all HTTP and HTTPS requests through OracleAS Web Cache. Ensure documents, especially HTTPS documents, are sent to authorized users through careful use of caching rules.

Depending on the application, you may or may not want requests for secure pages to go through the cache. If every HTTPS request is a non-cacheable page that is unique for each user session or is too sensitive for caching a copy, then route this traffic directly to the origin server. Because no traffic will be cached in this case, routing traffic to the origin server avoids extra encryption/decryption processing time at the OracleAS Web Cache layer.

5.1.3.2 Secure Administration, Invalidation, and Statistics Monitoring Using HTTPS

The default configuration for OracleAS Web Cache does not enable HTTPS for administration, invalidation, or statistics monitoring requests. Instead, these ports are configured for HTTP basic authentication. On an insecure network, the passwords for the `ias_admin`, `administrator`, and `invalidator` users can be decoded if they are sniffed out of the HTTP traffic. To avoid breach of security information for unprotected and insecure networks, set the communication protocol to HTTPS for administration and invalidation operations in the Operation Ports page (Ports > Operations Ports) of OracleAS Web Cache Manager.

5.1.3.3 Use Web Caching to Help Defend Against Denial-of-Service Attacks

OracleAS Web Cache was designed to provide high performance, reliability and scalability on low-cost commodity hardware. A single OracleAS Web Cache instance can be configured to support thousands of concurrent inbound HTTP connections. The throughput (requests/second) that a single cache instance can sustain scales linearly with CPU speed. Additionally, OracleAS Web Cache fully supports the HTTP/1.0 and HTTP/1.1 header fields, including `Keep-Alive`. The `Keep-Alive` header field

reduces the frequency of connection establishment and improves performance and scalability under heavy load.

When clustered, the capacity—the amount of content stored in RAM—of the cache tier scales linearly, as well. Cache clustering achieves high availability by failing over among cluster nodes, as well as high scalability by utilizing memory and processing power of multiple inexpensive computing hardware units in parallel.

Not surprisingly, many customers have reported the successful use of OracleAS Web Cache to prevent distributed and single-source denial-of-service attacks. Denial-of-service attacks attempt to prevent access to Web sites either by flooding sites with traffic volumes that surpass throughput and connection capacities or by sending malicious requests intended to exploit software flaws that cause servers to crash. By caching responses to denial-of-service requests, OracleAS Web Cache helps address the throughput and scalability limitations of Web sites, while creating a crucial barrier between malicious queries and a site origin application and database servers.

The guidelines for configuring connection limits, caching rules, and cache clusters are described in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.3.4 Change Passwords Frequently

You should change the administration and invalidation passwords for OracleAS Web Cache from the default passwords during or after installation. You should change the passwords on a regular basis, at least once a month.

Consider using a secure password that is at least 8 characters long. At least one of the characters must be a number.

5.1.4 Configuring OracleAS Web Cache

The following sections describe best practices related to configuring OracleAS Web Cache:

- Section 5.1.4.1, "Use the OracleAS Web Cache Manager to Avoid Configuration Problems"
- Section 5.1.4.2, "Configure Enough Memory"
- Section 5.1.4.3, "Allocate Sufficient Network Bandwidth"
- Section 5.1.4.4, "Set a Reasonable Number of Network Connections"
- Section 5.1.4.5, "Create Custom Error Pages"

5.1.4.1 Use the OracleAS Web Cache Manager to Avoid Configuration Problems

The OracleAS Web Cache Manager is a graphical user interface for administering, configuring, and managing caches. Oracle Corporation recommends that you use it rather than manually editing the configuration files to make configuration changes.

In cache clusters, it is especially important to use OracleAS Web Cache Manager to modify the configuration and to propagate it to the other cluster members. If you manually edit the configuration files, you may encounter problems.

For example, if you do not set the site name properly for each cluster member, identical objects could be cached in each cluster member, but the objects may be known by a different name in the different caches. For example, you could have the same object cached as:

```
mysite:7777:document1.html
myste:7777:document1.html
```

If the site names are not consistent, the objects with the variant site name will not be invalidated even if you use invalidation propagation.

5.1.4.2 Configure Enough Memory

To avoid swapping objects in and out of the cache, it is crucial to configure enough memory for the cache. Generally, the amount of memory (maximum cache size) for OracleAS Web Cache should be set to at least 256 MB.

To determine the maximum amount of memory required, take the following steps:

1. Determine what objects you want to cache, how many are smaller than 2 KB and how many are larger than 2 KB. Determine the average size of the objects that are larger than 2 KB. Determine the expected peak load—the maximum number of objects to be processed concurrently. Determine the average number of connections.
2. Calculate the amount of memory needed based on the number and size of the objects. The *Oracle Application Server Web Cache Administrator's Guide* provides a formula to use in calculating the amount of memory needed to cache your objects.

5.1.4.3 Allocate Sufficient Network Bandwidth

When you use OracleAS Web Cache, make sure that each node has sufficient network bandwidth to accommodate the throughput load. Otherwise, the network may be saturated even though OracleAS Web Cache has additional capacity. For example, if your application generates more than 100 megabits of data per second, 10/100 Megabit Ethernet will likely be saturated.

If the network is saturated, consider using Gigabit Ethernet rather than 10/100 Megabit Ethernet. Gigabit Ethernet provides the most efficient deployment scenario to avoid network collisions, retransmissions, and bandwidth starvations.

Additionally, consider using two separate network interface cards (NIC): one for incoming client requests and one for requests from the cache to the application Web server.

If system monitoring tools reveal that the network is under utilized and throughput is less than expected, check whether or not the CPUs are saturated.

5.1.4.4 Set a Reasonable Number of Network Connections

It is important to specify a reasonable number for the maximum connection limit for the OracleAS Web Cache server. If you set a number that is too high, performance can be affected, resulting in slower response time. If you set a number that is too low, fewer requests will be satisfied. You must strike a balance between response time and the number of requests processed concurrently.

For information about setting the number of network connections, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.4.5 Create Custom Error Pages

By default, OracleAS Web Cache ships with and is configured to serve the following error pages:

- `network_error.html`: This file is served when OracleAS Web Cache encounters network problems while connecting, sending, or receiving a response from an origin server for a cache miss request.

- `busy_error.html`: This file is served when origin server capacity has been reached.
- `esi_fragment_error.txt`. This file is served when OracleAS Web Cache is unable to fetch the src specified in an `<esi:include>` tag and the alt attribute, onerror attribute, or the try |attempt |except block are either not present or fail.

  The ESI language, which provides three specific elements for fine-grain control over content assembly in error scenarios:

  - The alt attribute of the `<esi:include>` tag
  - The onerror attribute of the `<esi:include>` tag
  - The try |attempt |except block
  - Default page in place of the fragment

  When the fragment specified for the src attribute of the `<esi:include>` tag cannot be fetched, the fragment specified with the alt attribute is used as an alternative. If the alt attribute is not used or the fragment cannot be fetched, the onerror attribute is used. The onerror attribute is used before the try |attempt |except block. If the try |attempt |except block does not exist, the exception handling is propagated to the parent or template page. If all the ESI language controls fail, OracleAS Web Cache displays a default page in place of the fragment.

For a production environment, Oracle Corporation advises that you modify the defaults or create entirely new error pages to be consistent with other error pages generated by your application.

For information on creating or modifying default error pages, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5 Increasing Cache Hits

The following sections provide tips in increasing the cache hit rate:

- Section 5.1.5.1, "Use Cookies and URL Parameters to Increase Cache Hit Ratios"
- Section 5.1.5.2, "Use Redirection to Cache Entry Pages"
- Section 5.1.5.3, "Use Surrogate-Control Headers Instead of Caching Rules"
- Section 5.1.5.4, "Use Partial Page Caching Where Possible"
- Section 5.1.5.5, "Use ESI Variables for Improved Cache Hit Ratio for Personalized Pages"
- Section 5.1.5.6, "Use the <esi:environment> Tag for Authentication or Authorization Callbacks"
- Section 5.1.5.7, "Use esi:inline and esi:include Tags Appropriately"
- Section 5.1.5.8, "Leverage JESI Over Hand-Generating the ESI Tags"

5.1.5.1 Use Cookies and URL Parameters to Increase Cache Hit Ratios

OracleAS Web Cache can cache different versions of a document with the same URL based on request cookies or headers. To use this feature, applications may need to implement some simple change, such as creating a cookie or header that differentiates the pages.

On the opposite side of the spectrum, some applications contain some insignificant URL parameters that lead to different URLs representing essentially the same content. If the documents are cached under their full URLs, then the cache-hit ratio becomes

very low. You can configure OracleAS Web Cache to ignore the non-differentiating URL parameter values when composing the "cache key" for documents, so a single document will be cached for different URLs, greatly increasing cache hit ratios.

Sometimes the content for a set of pages is nearly identical, but not exactly the same. For example, the pages may contain hyperlinks composed of the same URL parameters with different session-specific values, or they may include some personalized strings in the page text, such as welcome greetings and shopping cart totals. In this case, OracleAS Web Cache can still store one single copy of the document with placeholders for the embedded URL parameters and/or the personalized strings, and dynamically substitute the correct values into the placeholders when serving the document to clients.

You can also control whether a cached document can be served to a client based on its session state.

For more information on multiple version documents, sessions, ignoring URL parameter values, simple personalization, and how to control whether a cached document can be served to a request based on sessions, see Chapter 2, "Caching Concepts," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5.2 Use Redirection to Cache Entry Pages

For some popular site entry pages, such as "/", that typically require session establishment, session establishment effectively makes the page non-cacheable to all new users without a session.

To cache these pages while preserving session establishment, you can create a blank page that provides session establishment for all initial requests and redirects to the real popular page. This way, subsequent redirected requests to the popular page will carry the session, enabling the popular page to be served out of the cache.

For more information on configuring caching rules for pages requiring session establishment, see Chapter 9, "Creating Caching Rules," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5.3 Use Surrogate-Control Headers Instead of Caching Rules

There are two ways to specify the caching properties of an HTTP response using OracleAS Web Cache. You can use one or both of the following:

- Administrators can configure caching rules using the OracleAS Web Cache Manager interface.
- Application developers can set caching policies via the `Surrogate-Control` response header. If a given property is set in both a response header and the configuration, the value set by `Surrogate-Control` overrides rules specified in the configuration.

Although caching rules support the setting of more properties than the `Surrogate-Control` header, it is generally more manageable to set properties in the `Surrogate-Control` header whenever possible. For example, if you need to set the expiration policy and the multiple-version property for a document, it is preferable to use the `Surrogate-Control` header.

If you define many different categories of cacheable and non-cacheable documents, you need to carefully define rule selectors and rule priorities so that the appropriate rule is used for any document. Because a Surrogate-Control response header is only associated with one response and overrides the configuration, the properties set in Surrogate-Control will not be mistakenly replaced by other configuration rules or

newly created configuration rules. If you are creating new applications, consider building in Surrogate-Control response headers.

On the other hand, sometimes the configuration approach is more convenient. If a small number of rules are sufficient to describe all the caching properties of all documents that OracleAS Web Cache can receive from an origin server, then editing the configuration using the OracleAS Web Cache Manager administration interface may be simpler than generating Surrogate-Control headers for many documents.

Often, a combination of the two approaches is best.

5.1.5.4 Use Partial Page Caching Where Possible

Many Web pages, such as portal pages, are composed of fragments with unique caching properties. For these pages, full-page caching is not feasible. However, OracleAS Web Cache provides a partial page caching feature that enables each Web page to be divided into a template and multiple fragments that can, in turn, be further divided into templates and lower-level fragments.

Each fragment or template is stored and managed independently; a full page is assembled from the underlying fragments upon request. Fragments can be shared among different templates, so that common fragments are not duplicated to waste cache space. Sharing can also greatly reduce the number of updates required when fragments expire. Depending on the application, updating a fragment can be cheaper than updating a full page. In addition, each template or fragment may have its own unique caching policies such as expiration, validation, and invalidation, so that each fragment in a full Web page can be cached as long as possible, even when some fragments are not cached or are cached for a much shorter period of time.

For example, a Portal page may include stock quotes that expire in 20 minutes, news that expires in three hours, and rotating ad banners that should not be cached. To serve consistent content, traditional full-page caches need to update the entire page at the highest change frequency of all its fragments. With partial page caching, particular fragments can be updated, instead of the entire page.

OracleAS Web Cache uses Edge Side Includes (ESI) to achieve flexible partial-page caching. ESI is a simple markup language for partial-page caching. Applications can mark up HTTP responses with two different kinds of tags, `<esi:inline>` and `<esi:include>`, that define the fragment/template structure in the response.

5.1.5.5 Use ESI Variables for Improved Cache Hit Ratio for Personalized Pages

Personalized information often appears in Web pages, making them unique for each user. For example, many Web pages contain tens or hundreds of hyperlinks embedding application session IDs.

OracleAS Web Cache allows application developers to use variables in an ESI template. Because variables can be resolved to different pieces of request information or response information, the uniqueness of templates and fragments can be significantly reduced when personal information abounds.

There are two kinds of ESI variables: **request variables** and **response variables**. When an ESI template is assembled, a request variable is instantiated to a piece of request information such as a query string parameter, a cookie, or an HTTP header. For example, when a request for a dynamic page carries an application session ID in a query string parameter, this page may contain many hyperlinks with ESI request variables accessing this session ID, so that generated hyperlinks can carry the session ID into the next clicked page.

A response variable is similar to a request variable, except that its value comes not from the request, but from a special fragment called ESI environment. An **ESI environment** is a type of fragment with a response that defines a set of variables that can be accessed by response variable occurrences in the enclosing template. The tag itself does not contribute to the final assembled output. For example, a dynamic page with a calendar may need to present personal appointments that cannot be stored in Web browser cookies due to cookie size limits. The application can instead refer to a "profile" environment fragment in the template, in effect referring to all appointments in the environment without making separate requests for each appointment. In addition, you can merge multiple small fragments into one environment, so that each fragment can be referenced through response variable instantiation. This reduces storage and retrieval overhead similarly.

5.1.5.6 Use the <esi:environment> Tag for Authentication or Authorization Callbacks

Some applications protect certain Web pages with authentication, authorization, or session validation in the HTTP request. Even though the page content can be cached, every HTTP request must be authenticated, authorized, or validated by the origin server. For these pages, it is not appropriate to cache the full page. While it is possible to utilize JavaScript to achieve authentication, authorization, and validation through a separate HTTP request, the `<esi:environment>` tag provides a better solution to this problem.

If a page has cacheable content but requires mandatory authentication, authorization, and session validation, you can enclose an `<esi:environment>` tag in the page referencing a non-cacheable environment, and cache the enclosing page. When the cached page is requested, an HTTP request that specifies the environment will always be sent to the origin server, making a callback to the application. In this callback request, if you want to validate a cookie in the enclosing page request for session validation, authorization, or authentication, specify the `<esi:environment>` tag to include that cookie in its request. You can also include other information from the page request in the environment request.

If authentication, authorization, and validation are passed, your application should return HTTP status code 200 and any ESI environment response. OracleAS Web Cache proceeds to finish assembling the page. If the authentication, authorization, or validation fail, your application should return an appropriate HTTP status code (at or above 500 to denote a server error, or between 400 and 499 to denote a client request error). Then, OracleAS Web Cache will recognize that this environment has failed, and resort to standard ESI exception handling to abort this page or output an alternative error page or login page.

For more information about using the `<esi:environment>` tag or implementing ESI exception handling, see Chapter 15, "Edge Side Includes (ESI) Language Tags," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.5.7 Use esi:inline and esi:include Tags Appropriately

The `<esi:inline>` and `<esi:include>` tags enable applications to adopt ESI page fragmentation and assembly.

If a partial page fragment can be fetched independently, such as with a `URLPortlet`, refer to it with an `<esi:include>` tag. This enables OracleAS Web Cache to fetch the fragment independently of the rest of the page or fragments. However, if the fragment cannot be generated independently of the surrounding page, use the `<esi:inline>` tag, which enables OracleAS Web Cache to cache the inline fragment and re-use it in different contexts.

5.1.5.7.1 Using esi:inline for Non-Fetchable Fragmentation Most existing applications are designed to output an entire Web page to HTTP requests. These fragments and templates are **non-fetchable**, meaning they are not to be fetched independently from the origin server. If a cache needs any of these fragments or templates, the corresponding full Web page must be requested. To use ESI page assembly for non-fetchable fragments, an application can output the full-page response just as it does normally, with the exception that at the beginning and the end of each fragment, an `<esi:inline>` tag is inserted with a fragment name to demarcate the fragment. OracleAS Web Cache stores the enclosed portions as separate fragments and the original page as page templates without the enclosed fragments. Fragments are shared if their names are identical.

When an application uses non-fetchable `<esi:inline>` fragments, the full page must be requested for every cache miss. At first, it can appear that there is no apparent cache benefit for cache misses. However, non-fetchable `<esi:inline>` fragments improve overall caching by:

- Increasing the cache-hit ratio because shared cacheable fragments can be extracted and stored only once, reducing the size of the dynamic portion. A reduced space requirement results in a higher cache-hit ratio than full page caching.
- Reducing cache update frequency. Dynamic shared cacheable fragments require only one update. For example, a shared stock market fragment may expire much more frequently than any other parts of the page. With `<esi:inline>` fragmentation, only one cache update of any full page containing this fragment is enough to bring all full pages sharing this fragment current. Therefore, even non-fetchable `<esi:inline>` fragments can significantly reduce cache update frequency. The cost reduction is proportional to the degree of sharing.

The `<esi:inline>` tag is primarily intended for pages with cacheable fragments. If a page contains non-cacheable, non-fetchable fragments, then the use of `<esi:inline>` is not recommended. However, the update of this full page may still offer benefit if it contains some cacheable fragments that are shared with other pages.

5.1.5.7.2 Using esi:include for Fetchable Fragmentation The `<esi:include>` tag is another way to define fragments and templates in an HTTP output for dynamic content caching and assembly. The page including an `<esi:include>` tag is a template that refers to the defined fragment. However, because it has some key differences from `<esi:inline>`, its applicable scenarios are very different:

- An `<esi:include>` tag in a template only defines the reference to a fragment. It does not enclose an embedded fragment directly in the template.
- A fragment referenced by an `<esi:include>` tag must always be independently fetchable by HTTP or HTTPS.

  The requested URL is the same as the fragment name. In contrast, an `<esi:inline>` tag name only identifies the uniqueness of the fragment and is not used to fetch the actual content. The attribute defining the fragment name in `<esi:include>` fragment is src instead of name.

There are at least two scenarios where using `<esi:include>` tags is beneficial:

- Some applications, such as a Web portal, assemble content from external sources. The application only provides a template that is used to fetch various fragments from third-party sources. In this case, the `<esi:include>` tags fetch and assemble directly, reducing one layer of redundancy.
- Some applications offer faster responses for template-only or fragment-only requests than full-page requests that use `<esi:inline>` tags. If `<esi:include>`

is used for page fragmentation and assembly, OracleAS Web Cache can miss only on the templates or fragments when most or all fragments are already cached, saving effective cache miss cost. In many cases, it is also valuable to cache the personalized templates because these seldom change.

5.1.5.8 Leverage JESI Over Hand-Generating the ESI Tags

In dynamic applications, you can use ESI tags for better caching in two ways: hand generating the tags, or using Edge Side Includes for Java (JESI) tags.

The JESI specification is a specification and custom JSP tag library that developers can use to automatically generate ESI code. JESI facilitates the programming of Java ServerPages (JSPs) using ESI. While developers can always use ESI tags directly within their JSP code, JESI represents an easy way to express the modularity of JSPs and the caching of those modules, without requiring developers to learn a new programming syntax. JESI generates the appropriate ESI tags and headers in the JSP output that instruct ESI processors, such as OracleAS Web Cache, to cache (or not) templates and fragments for the appropriate duration. JESI also facilitates the partial execution of JSPs when an ESI processor requests fragments and templates.

5.1.6 Invalidation and Expiration

The following sections provide tips about invalidating and expiring cache objects:

- Section 5.1.6.1, "Use Basic Invalidation for Single Objects"
- Section 5.1.6.2, "Use Substring Matching for Multiple Objects in Advanced Invalidations"
- Section 5.1.6.3, "Build Programmatic Invalidation Into Application Logic"
- Section 5.1.6.4, "Combine Invalidation and Expiration Policies"
- Section 5.1.6.5, "Use Invalidation Propagation in Clusters and Hierarchies"
- Section 5.1.6.6, "Tune Invalidation Performance Using Indexes"

5.1.6.1 Use Basic Invalidation for Single Objects

When you need to invalidate one object in the cache, send a basic rather than an advanced invalidation request to avoid cache traversal. Advanced invalidation requests should be reserved for invalidation of multiple objects.

To send a basic invalidation request, use the Basic Content Invalidation page (Operations > Basic Content Invalidation) in OracleAS Web Cache Manager or specify the BASICSELECTOR element in a manual invalidation request.

For example, the following request invalidates a document exactly matching /contacts/contacts.html using the BASICSELECTOR element:

```
<?xml version="1.0"?>
<!DOCTYPE INVALIDATION SYSTEM
"internal://WCSinvalidation.dtd">
<INVALIDATION VERSION="WCS-1.1">
    <OBJECT>
      <BASICSELECTOR URI="http://www.company.com:80/contacts/contacts.html"/>
      <ACTION REMOVALTTL="0"/>
    </OBJECT>
</INVALIDATION>
```

5.1.6.2 Use Substring Matching for Multiple Objects in Advanced Invalidations

When multiple objects share a common URL, request POST body, or an embedded URL parameter, you can express the common elements in multiple ways:

- Common URL:
  - Use the **URL Expression** field in the Advanced Content Invalidation page.
  - Use the URIEXP attribute of the ADVANCEDSELECTOR element.
  - Use the ADVANCEDSELECTOR element OTHER element with a NAME attribute value of URL.
- Common Request POST Body:
  - Use the **HTTP Method** and **POST Body Expression** fields in the Advanced Content Invalidation page
  - Use the ADVANCEDSELECTOR element METHOD and BODYEXP attributes
  - Use the ADVANCEDSELECTOR element OTHER element with a NAME attribute value of BODY.
- Common Embedded URL Parameter:
  - Use the ADVANCEDSELECTOR element OTHER element with a NAME attribute value of QUERYSTRING_PARAMETER.

For the quickest invalidation, Oracle Corporation recommends using the OTHER element to specify a substring for literal matching rather than regular expression for pattern matching.

To send an advanced invalidation request, use the Advanced Content Invalidation page (Operations > Advanced Content Invalidation) in OracleAS Web Cache Manager or specify the ADVANCEDSELECTOR element in a manual invalidation request.

To send an advanced invalidation request with substring matching:

1. Specify the OTHER element in a manual invalidation request that uses the ADVANCEDSELECTOR element.
2. Specify the NAME attribute to use a value of URL, BODY, or QUERYSTRING_PARAMETER.
3. Specify the TYPE attribute to use a value of SUBSTRING.

For example, the request in the following example searches for any documents below http://wc-cluster.us.oracle.com:1100/pls/portal/!PORTAL.wwpro_app_provider.execute_portlet/595897563/, that match the following criteria:

- The HTTP request method is an HTTP POST request method.
- The URI is `showPortlet.Show`.
- The HTTP requests headers are `x-oracle-cache-user` and `x-oracle-cache-subid`.
- The embedded URL parameters are `_portlet_id` and `_provider_id`.

```
<?xml version="1.0"?>
<!DOCTYPE INVALIDATION SYSTEM "http://www.oracle.com/webcache/90200/WCSinvalidation.dtd">
<INVALIDATION VERSION="WCS-1.1">
  <OBJECT>
    <ADVANCEDSELECTOR
      URIEXP="/pls/portal/!PORTAL.wwpro_app_provider.execute_portlet/595897563/"
      HOST="wc-cluster.us.oracle.com:1100" METHOD="POST">
      <OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
```

```
            VALUE="_portlet_id=2"/>
        <OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
            VALUE="_provider_id=595897563"/>
        <HEADER NAME="x-oracle-cache-user" VALUE="PORTAL"/>
        <HEADER NAME="x-oracle-cache-subid" VALUE="1"/>
        <OTHER NAME="BODY" TYPE="SUBSTRING" VALUE="_language=EN-US"/>
        <OTHER NAME="URI" TYPE="SUBSTRING" VALUE="showPortlet.Show"/>
      </ADVANCEDSELECTOR>
  <ACTION REMOVALTTL="0"/>
<INFO VALUE="Invalidate an old portlet based on portlet ID and provider ID"/>
</OBJECT>
</INVALIDATION>
```

5.1.6.3 Build Programmatic Invalidation into Application Logic

OracleAS Web Cache is designed for caching highly dynamic Web pages. There are several ways to safeguard the correctness of the cached content. For those cached pages with content that changes following unpredictable actions by Web site users, the most effective way to ensure correct content is to build programmatic invalidation into application logic.

The application Web server and database are two areas that may benefit from embedded programmatic invalidation. On the application Web server, you can build invalidation into CGIs, JSPs and Java servlets using the OracleAS Web Cache Java invalidation API, `jawc.jar`. The `jawc.jar` file is located in the *$ORACLE_HOME*`/webcache/jlib` directory on UNIX and *ORACLE_HOME*`\webcache\jlib` directory on Windows. For example, page A displays information about a certain bike in stock. This page is cacheable. Page B provides users a way to reserve one bike for purchase. On the mid-tier, there is a Java servlet or JSP to service page B. To make this servlet cache-aware, use `jawc.jar` to invalidate page A.

Similarly, you can build invalidation into PL/SQL applications using the OracleAS Web Cache PL/SQL invalidation API `wcvutil.sql` and `wcvappl.sql`. This way, developers can embed the invocation of invalidation PL/SQL procedure calls into the PL/SQL Web page. The APIs, `wcvutil.sql` and `wcvappl.sql`, are located in the *$ORACLE_HOME*`/webcache/examples` directory on UNIX and *ORACLE_HOME*`\webcache\examples` directory on Windows.

To facilitate the caching of JSPs, developers can use the JESI custom tag library included with OC4J and Oracle JDeveloper. One of the tags, `<jesi:invalidate>`, enables programmatic invalidation when the JSP engine processes a page containing this tag. For more information about JESI, see the *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*.

Alternatively, developers can tie invalidation logic to database updates. In the same bike example, you can use a PL/SQL invalidation procedure call to invalidate pages that rely on inventory data stored in the database. In other words, you can apply a database trigger to the row or table containing information about the bike.

For more information about invalidation, see Chapter 11, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.6.4 Combine Invalidation and Expiration Policies

With expiration rules, cached objects are marked as invalid after a certain amount of time in the cache. Expirations are useful if it can be accurately predicated when content will change on an origin server or database. To prevent documents from remaining in the cache indefinitely, Oracle Corporation recommends creating expiration rules for all cached documents.

With invalidation requests, an HTTP POST message specifies which objects to mark as invalid. Invalidation requests are intended for less predictable, more frequently changing content. Send invalidation requests when you know objects have been refreshed on the origin server. Invalidation policies can be automated, as described in Section 5.1.6.3.

5.1.6.5 Use Invalidation Propagation in Clusters and Hierarchies

In a cache cluster or hierarchy, you can send invalidation messages to one cache, and that cache propagates the invalidation messages to the other caches in the cluster or hierarchy.

The benefits of invalidation propagation include data consistency across cluster members and ease of use for the administrator.

However, under the following circumstances, you may want to disable invalidation propagation and send the invalidation messages to each individual member of the cluster:

- When the cluster membership is in flux. For example, as you begin deployment, you may have made changes to cluster members but have not yet propagated the configuration changes to all members. In this case, the invalidation messages are not propagated to the members with different configurations.
- If you do not want to invalidate data for all cluster members. For example, because of time zone differences, you want to send invalidation messages to only some of the cluster members at one time.

Note that if you do not invalidate data for all cluster members, the cached data may become inconsistent. In addition, cluster members may serve stale data, not only in response to requests from clients, but also in response to requests from their peers.

For more information about invalidation propagation, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.6.6 Tune Invalidation Performance Using Indexes

To improve performance of advanced invalidation requests that use QUERYSTRING_PARAMETER to match objects with the same embedded URL parameters, manually create an invalidation index. Because an invalidation index creates more depth to an internal invalidation tree used by OracleAS Web Cache, OracleAS Web Cache is able to categorize objects in the cache for faster lookup and traversal. To use this feature for a URL or a request POST body, consider moving the URI or BODY value to QUERYSTRING_PARAMETER instead.

To specify an invalidation index, add an INVALIDATIONINDEX element to the webcache.xml file that specifies the substring value used for QUERYSTRING_PARAMETER:

```
<SECURITY>
...
</SECURITY>
<INVALIDATIONINDEX>
    <INDEXPARAM VALUE="VALUE_of_QUERYSTRING_PARAMETER"/>
    <INDEXPARAM VALUE="VALUE_of_QUERYSTRING_PARAMETER"/>
</INVALIDATIONINDEX>
<WATCHDOG ENABLE="YES|NO"/>
```

Oracle Corporation recommends using invalidation indexes to create more depth to flat directory structures.

For more information about invalidation, see Chapter 11, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.7 Optimizing Response Times

The following sections describe best practices in optimizing response times:

- Section 5.1.7.1, "Optimize Response Time By Tuning Origin Server and OracleAS Web Cache Settings"
- Section 5.1.7.2, "Improve Response Times and Reduce Network Bandwidth With Compression"
- Section 5.1.7.3, "Use Only Warning or Notification Logging Levels to Conserve Resources"

5.1.7.1 Optimize Response Time By Tuning Origin Server and OracleAS Web Cache Settings

If you have not configured the origin server or the cache correctly, response time may be slower than anticipated. If the origin server is responding more slowly than expected or if the origin server is not responding to requests from the cache because it has reached its capacity, check the origin server and the OracleAS Web Cache settings.

First, check the values of the following settings in the application Web server configuration file (`httpd.conf`). (These particular parameter names are specific to the Oracle HTTP Server.)

- `KeepAlive`: Whether to allow persistent connections. Persistent connections allow a client to send multiple sequential requests through the same connection.

  Make sure `KeepAlive` is enabled. This can improve performance because the connection is set up only once and is kept open for subsequent requests from the same client.

- `KeepAliveTimeout`: The time a connection is left open to wait for the next request from the same client. If requests are primarily from OracleAS Web Cache, you can set this value fairly high. A reasonable value is 30 seconds.

- `MaxKeepAliveRequests`: The maximum number of requests to allow during a persistent connection. Set to 0 to allow an unlimited number of requests.

- `MaxClients`: The maximum number of clients that can connect to the application Web server simultaneously.

  If `KeepAlive` is enabled for the application Web server, you may require more concurrent httpd server processes, and you may need to set the `MaxClients` directive to a higher value.

  If client requests have a short response time, you may be able to improve performance by setting `MaxClients` to a lower value. However, when this value is reached, no additional processes will be created, causing other requests to fail.

  The `MaxClients` limit on the application Web server should be greater than or equal to the application Web server capacity as set through the OracleAS Web Cache Manager.

  See the *Oracle Application Server 10g Performance Guide* for more information about setting limits for the origin server.

Then, if the origin server is still busier than anticipated, it may mean that the cache cannot process the requests and is routing more requests to the origin server. Check the following OracleAS Web Cache settings:

- The origin server capacity, as set through the OracleAS Web Cache Manager. See Chapter 7, "Basic Setup and Configuration," in the *Oracle Application Server Web Cache Administrator's Guide* for information about setting origin server capacity.
- The number of cache connections (Maximum Incoming Connections) in the Resource Limits page (Properties > Resource Limits) of the OracleAS Web Cache Manager. If you have specify too low of a limit and the cache reaches the limit, OracleAS Web Cache may drop connections.
- The memory size for the cache (Maximum Cache Size) in the Resource Limits page. See Chapter 7, "Basic Setup and Configuration," in the *Oracle Application Server Web Cache Administrator's Guide* for information about calculating the memory size.
- The cache cluster capacity. In a cluster, if cluster capacity is too low, a cache may not receive a response for owned content from a peer cache within the specified interval. As a result, the request is sent to the origin server.
- Keep-Alive Timeout in the Network Timeouts page (Properties > Network Timeouts): This is the amount of time a network connection is left open after OracleAS Web Cache sends a response to a Web browser. Keep-Alive allows an HTTP client to send multiple requests to OracleAS Web Cache using the same network connection. By default, the connection is left open for five seconds, which is typically enough time for the Web browser to send subsequent requests to OracleAS Web Cache using the same connection.

  If the network between the Web browser and OracleAS Web Cache is slow, consider increasing the timeout, perhaps up to 30 seconds.
- Origin Server Timeout in the Network Timeouts page (Properties > Network Timeouts): This is the amount of time for the application Web server to generate a response to OracleAS Web Cache. If the application Web server or proxy server is unable to generate a response within that time, OracleAS Web Cache sends a network apology page to the Web browser.

  Usually, this value should be equal to the response time of the slowest document served by the application Web Server. If the value is too low, long-running requests will timeout before the response is complete. If the value is too high and the application Web server hangs for some reason, it will take longer for OracleAS Web Cache to failover to another application Web server.

For information on specifying these settings, see the *Oracle Application Server Web Cache Administrator's Guide*.

If these resources are set reasonably, check the following:

- Caching rules. Make sure that you are caching the appropriate objects including popular objects.
- Priority rankings of the caching rules. Give the non-cacheable documents a higher priority than the cacheable documents.
- The number of rules. If you have a large number of rules, parsing of rules will take additional time.

If the settings for the origin server and OracleAS Web Cache are set correctly, but the response times are still higher than expected, check system resources, especially:

- Network bandwidth. See Section 5.1.4.3, "Allocate Sufficient Network Bandwidth"
- CPU usage. See Section 5.1.2.1, "Use Two CPUs and Consider Deploying on Dedicated Hardware"

- The TCP time-wait setting. This setting controls the amount of time that the operating system holds a port, not allowing new connections to use the same port. See the *Oracle Application Server 10g Performance Guide* for more information.

5.1.7.2 Improve Response Times and Reduce Network Bandwidth With Compression

OracleAS Web Cache features automatic compression of dynamically generated content. On average, using the standard GZIP algorithm, OracleAS Web Cache is able to compress text files, such as HTML and XML by a factor of four. Because compressed objects are smaller in size, they require less bandwidth to transmit and can be delivered faster to Web browsers. With compression, everyone benefits: Internet Service Providers (ISPs), Hosting Service Provider (HSPs), corporate networks and content providers reduce their transmission costs, while end-users enjoy more rapid response times. Since 1997, all major Web browsers support the expansion of GZIP encoded documents.

Most application Web servers on the market are capable of serving compressed pages, but few enable caching of compressed output. With OracleAS Web Cache, compression is a simple Yes/No option that an administrator selects when specifying a caching rule. Because OracleAS Web Cache supports regular expression for caching rules, compression can be applied to responses using criteria other than just file extension. Regular expression makes it very easy to select which pages to compress and which pages not to compress, as well as whether or not a particular Web browser should receive compressed content. Unlike the typical application Web server, OracleAS Web Cache offers compression and caching for pages that have been dynamically generated and for ESI fragments specified in a Surrogate-Control response header. By caching compressed output, OracleAS Web Cache reduces the processing burden on the application Web server, which would otherwise have to re-generate and compress dynamic pages each time they are requested. Because compressed objects are smaller in size, they are delivered faster to Web browsers with fewer round-trips, reducing overall latency. In addition, compressed objects consume less cache memory.

Do not compress images, such as GIFs and JPEGs, as well as executables and files that are already zipped with utilities like WinZip and GZIP. Compressing these files incurs additional overhead without the benefits of compression. Also, do not compress JavaScript includes (.js) and Cascading Style Sheets (.css), as some Web browsers have difficulty expanding these file types.

For more information on compression, see the *Oracle Application Server Web Cache Administrator's Guide*.

5.1.7.3 Use Only Warning or Notification Logging Levels to Conserve Resources

You can specify the level of detail for the event log. The Trace or Debug levels are useful for debugging purposes. However, using either of these levels of event logging consumes system resources. For example, the log file might fill up disk space, causing OracleAS Web Cache to crash. Unless you need to diagnose problems, you should the Warning or Notification levels. With these levels, OracleAS Web Cache writes only typical events to the event log. Also, consider disabling access logging unless you are monitoring end-user performance and access.

For information on specifying these settings, see Chapter 12, "Logging Events, Diagnostics, and Access Information" in the *Oracle Application Server Web Cache Administrator's Guide*.

ORCL00384237