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 MEMORANDUM IN OPPOSITION TO ORACLE'S MOTION FOR SUMMARY JUDGMENT OF NONINFRINGEMENT

VOLUME II OF II

EXHIBITS 18-29

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
880382 / 31393 / Oracle

TABLE OF CONTENTS

Exh.	*Document*
1.	U.S. Patent 5,894,554, Lowery et al.
2.	U.S. Patent 6,415,335, Lowery et al.
3.	Oracle's Proposed Claim Construction Chart, Dec. 14, 2007
4.	Order, Aug. 7, 2008, *EpicRealm Licensing, LLC v. Autoflex Leasing, Inc. et al.,* (E. D. Tex.)
5.	Excerpt from deposition transcript of Mark Clark, April 17, 2008
6.	Excerpt from deposition transcript of Paul C. Clark, July 11, 2008
7.	Excerpt from deposition transcript of David Finkel, June 20, 2008
8.	Excerpt from deposition transcript of Steve Harris, April 18, 2008
9.	Excerpt from deposition transcript of Michael Shamos, June 30, 2008
10.	Excerpt from deposition transcript of Douglas Surber, Feb. 28, 2008
11.	Excerpt from Expert Report Dr. Paul C. Clark, June 6, 2008
12.	Excerpt from Rebuttal Expert Report of David Finkel, June 6, 2006
13.	Excerpt from Expert Report Shamos
14.	Excerpt from Oracle9i Application Server Performance Guide Release 2 (9.0.2) April 2002, A95102-02, ORCL00847081-00847294
15.	Excerpt from Oracle Application Server Containers for J2EE 10g Release 2 (10.1.2), B14014-02, ORCL00444343-00444558
16.	Excerpt from Oracle Call Interface Programmer's Guide 11g Release 1 (11.1) B28395-03, May 2008
17.	Mod_oc4j Load Balancing in iAS 9.0.4, ORCL00018115-00018150
18.	Design Specification for OCI Runtime Connection Load Balancing in 11g, ORCL01310468-01310498
19.	Test Specification for Application Runtime and Recovery Performance using Runtime Load Balancing, Oracle 10gR2, ORCL01383645-01383660
20.	Oracle World, ORCL01050220-01050280
21.	Excerpt from Oracle Application Server 10g, Best Practices 10g (9.0.4) B12223-01, ORCL00384158-00384305

22. Oracle9iAS Web Cache Administrative and Deployment Guide, Release 2 (9.0.2) May 2002, A95404-02, ORCL00272342-00272887

23. Email from R. Vairavaraja, 8/24/07, ORCL01436981

24. Letter from J. Moore, FreshDirect, 3/11/08, EPIC436363

25. Excerpt from Oracle Application Server Best Practices Guide 10g Release 2 (10.1.2), B28654-01, March 2006, ORCL00528055-00528218

26. USSN 08/794,269, Office Action, 8/23/2000, ORCL01623264-01623276

27. USSN 09/615,383, Information Disclosure Statement, Dec. 12, 2005

28. Excerpt from Oracle Web Cache Administration and Deployment Guide, Release 1.0.2, A86722-01, Nov. 21, 2000, ORCL00107955-00108126

29. USSN 08/962,536, Information Disclosure Statement, Nov. 21, 2000 ORCL01622918-01622919

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 18

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 19

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 20

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 21

Oracle® Application Server 10*g*

Best Practices

10*g* (9.0.4)

Part No. B12223-01

May 2004

ORACLE

Oracle Application Server 10*g* Best Practices, 10*g* (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 10*g*, 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).

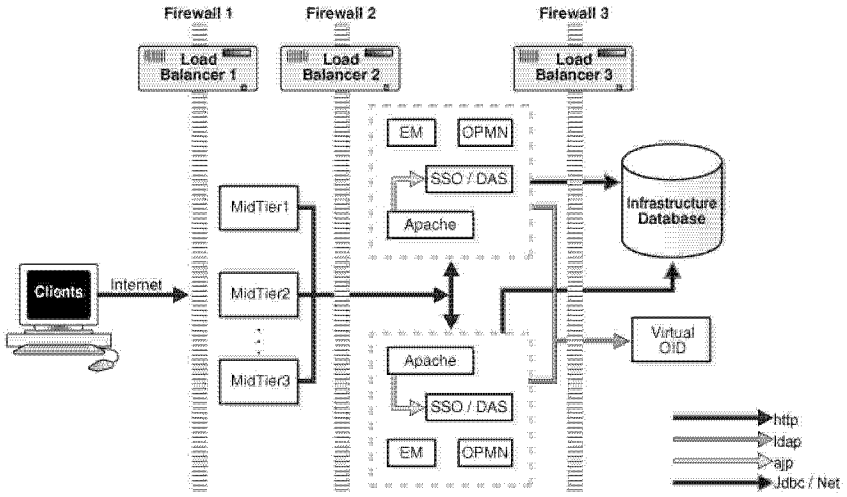
AFC refers to the deployment of Oracle Identity Management and the OracleAS Metadata Repository on hardware clusters where each node of the cluster runs the database instances as well as the ID processes and all of them access a shared RAC database as their repository.

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

ORCL00384215

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 10*g* is the metric based policy. When this type of policy is specified in the mod_oc4j.conf file, mod_oc4j routes

ORCL00384216

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*.

ORCL00384217

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.

ORCL00384219

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

ORCL00384220

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

ORCL00384221

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

ORCL00384222

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
```


ORCL00384223

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.

ORCL00384224

- `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

ORCL00384225

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

ORCL00384226

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.

ORCL00384227

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.

ORCL00384228

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>`

ORCL00384229

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>
```


ORCL00384230

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 URI.
- 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 URI, 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
      URIPREFIX="/pls/portal/!PORTAL.wwpro_app_provider.execute_portl et/595897563/"
      HOST="wc-cluster.us.oracle.com:1100" METHOD="POST">
      <OTHER NAME="QUERYSTRING_PARAMETER" TYPE="SUBSTRING"
```


ORCL00384231

```
            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 `wxvutil.sql` and `wxvappl.sql`. This way, developers can embed the invocation of invalidation PL/SQL procedure calls into the PL/SQL Web page. The APIs, `wxvutil.sql` and `wxvappl.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.

ORCL00384232

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.

ORCL00384233

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:

ORCL00384234

- 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"

ORCL00384235

- 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*.

ORCL00384236

ORCL00384237

Exhibit 22

Oracle9*i*AS Web Cache

Administration and Deployment Guide

Release 2 (9.0.2)

May 2002

Part No. A95404-02

ORACLE

Oracle9*i*AS Web Cache Administration and Deployment Guide, Release 2 (9.0.2)

Part No. A95404-02

Copyright © 2002 Oracle Corporation. All rights reserved.

Primary Author: Deborah Steiner

Contributing Author: Helen Grembowicz

Graphic Designer: Valarie Moore

Contributors: Marcos Almeida, Jesse Anton, Christine Chan, Bikui Chen, Rishi Divate, Ajay Dsai, Joe Errede, Jay Feenan, Patrick Fry, Hideaki Hayashi, Larry Jacobs, Lars Klevan, Wei Lin, Gary Ling, Martin Littlecott, Xiang Liu, Rajiv Mishra, Jordan Parker, Marcin Porwit, Charles Qi, Cyril Scott, Jiangping Shi, Parthiban Thilagar, Bill Wright, Jean Zeng,Tie Zhong, and Yuhui Zhu

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; 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. Oracle Corporation does not warrant that this document is 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, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions 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 Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark, and Oracle Store, Oracle8*i*, Oracle9*i*, Oracle9*i*AS Discoverer, PL/SQL, and SQL*Plus are trademarks or registered trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

1

Introduction to Oracle9*i*AS Web Cache

This chapter describes the performance barriers faced by Web sites and introduces the technology that can provide a complete caching solution.

This chapter contains these topics:

- What is the Big Picture for Caching?
- Oracle's Solution to Web Site Performance Issues
- How Web Caching Works
- Benefits of Web Caching
- Features of Oracle9iAS Web Cache
- Compatibility with Oracle9iAS Components

What is the Big Picture for Caching?

The electronic business model creates new performance requirements for Web sites. To carry out e-business successfully, Web sites must protect against poor response time and system outages caused by peak loads. Slow performance translates into lost revenue.

Many high-volume Web sites try to counter this problem by adding more application Web servers to their existing architecture. As more users access these Web sites, more and more application Web servers will have to be added. In short, the manageability costs associated with adding application Web servers often outweigh the benefits.

Static caches and content distribution services can provide some relief. However, these solutions are unable to serve content that is dynamically generated.

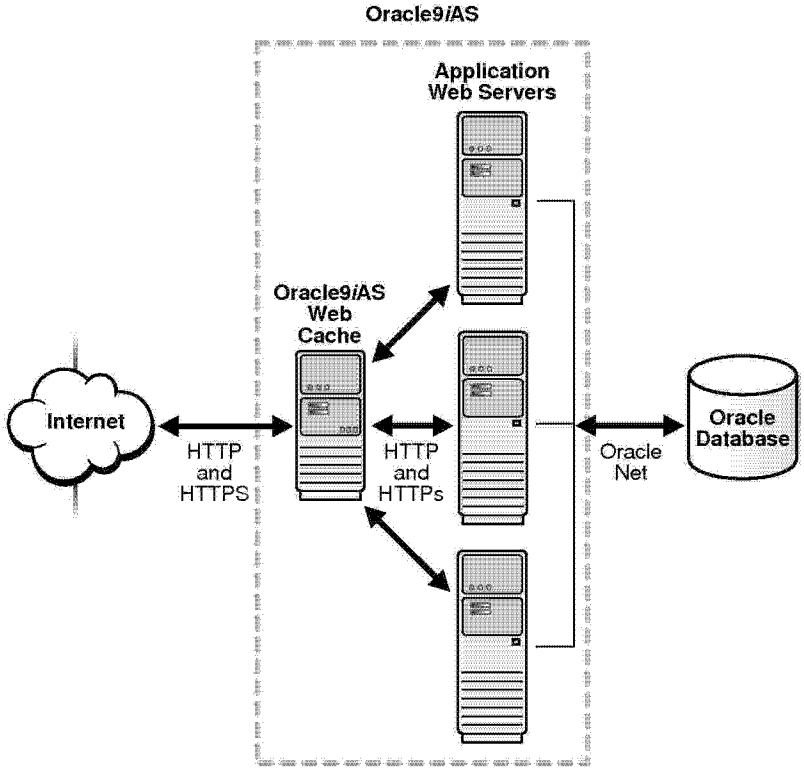
Oracle's Solution to Web Site Performance Issues

Faced with these performance challenges, e-businesses need to invest in more cost-effective technologies and services to improve the performance of their sites. Oracle offers Oracle9*i*AS Web Cache to help e-businesses manage Web site performance issues. Oracle9*i*AS Web Cache is a content-aware server accelerator, or reverse proxy server, that improves the performance, scalability, and availability of Web sites that run on Oracle9*i* Application Server (Oracle9*i*AS) and Oracle9*i*.

By storing frequently accessed URLs in memory, Oracle9*i*AS Web Cache eliminates the need to repeatedly process requests for those URLs on the application Web server. Unlike legacy proxies that handle only static documents, Oracle9*i*AS Web Cache caches both static and dynamically generated content from one or more application Web servers. Because Oracle9*i*AS Web Cache is able to cache more content than legacy proxies, it provides optimal performance by greatly reducing the load on application Web servers.

Figure 1–1 shows the basic architecture. Oracle9*i*AS Web Cache sits in front of application Web servers, caching their content, and providing that content to Web browsers that request it. When Web browsers access the Web site, they send HTTP protocol or HTTPS protocol requests to Oracle9*i*AS Web Cache. Oracle9*i*AS Web Cache, in turn, acts as a virtual server to the application Web servers. If the requested content has changed, Oracle9*i*AS Web Cache retrieves the new content from the application Web servers. The application Web servers may retrieve their content from an Oracle database.

Figure 1–1 Oracle9iAS Web Cache Architecture



Note: Oracle9*i*AS Web Cache is compatible with Oracle HTTP Server or any other HTTP-compliant application Web server.

How Web Caching Works

To Web browsers, Oracle9*i*AS Web Cache acts as the virtual server for application Web servers. You configure a Load Balancer with the same IP address that is registered for a site's domain name and the application Web servers' host names. This Load Balancer receives requests for Oracle9*i*AS Web Cache. This configuration enables Web browsers to communicate with Oracle9*i*AS Web Cache rather than application Web servers when accessing a Web site.

Figure 1–2 on page 1-5 shows how Web caching works. Oracle9*i*AS Web Cache has an IP address of `144.25.190.240` and the application Web server has an IP address of `144.25.190.245`. The steps for browser interaction with Oracle9*i*AS Web Cache follow:

1. A browser sends a request to a Web site named `www.company.com:80`.

   This request in turn generates a request to Domain Name System (DNS) for the IP address of the Web site.

2. DNS returns the IP address of the Load Balancer for the site, that is, `144.25.190.240`.

3. The browser sends the request for a Web page to the Load Balancer. In turn, the Load Balancer sends the request to Oracle9*i*AS Web Cache server `144.25.190.241`.

4. If the requested content is in its cache, then Oracle9*i*AS Web Cache sends the content directly to the browser. This is called a **cache hit**.

   Note: Dynamic content is generated by the application Web server and then returned to Oracle9*i*AS Web Cache before being passed to the browser.

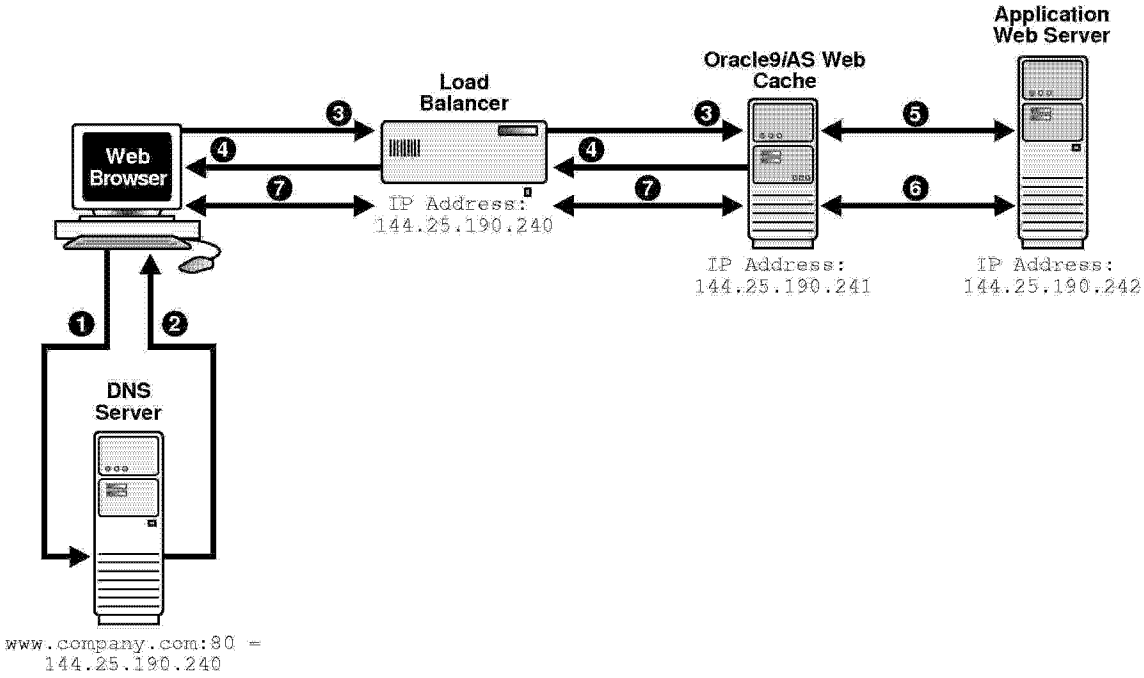
5. If Oracle9*i*AS Web Cache does not have the requested content or the content is stale or invalid, it hands the request off to application Web server `144.25.190.242`. This is called a **cache miss**.

6. The application Web server sends the content to Oracle9*i*AS Web Cache.

7. Oracle9*i*AS Web Cache sends the content to the client and makes a copy of the page in cache.

   Note: A page stored in the cache is removed when it becomes invalid or outdated, as described in "Cache Freshness and Performance Assurance" on page 2-4.

Figure 1–2 Web Server Acceleration



Benefits of Web Caching

Web caching provides the following benefits for Web sites:

- Performance

  Running on inexpensive hardware, Oracle9*i*AS Web Cache can increase the throughput of a Web site by several orders of magnitude. In addition, Oracle9*i*AS Web Cache significantly reduces response time to browser requests by storing documents in memory and by serving compressed versions of documents to browsers that support the GZIP encoding method.

- Scalability

  In addition to unparalleled throughput, Oracle9*i*AS Web Cache can sustain thousands of concurrent browser connections, meaning that visitors to a site see fewer application Web server errors, even during periods of peak load.

- High Availability

  Oracle9*i*AS Web Cache supports content-aware load balancing and failover detection. These features ensure that cache misses are directed to the most available, highest-performing Web server in the cluster. Moreover, a patent-pending capacity heuristic guarantees performance and provides surge protection when the application Web server load increases.

- Cost Savings

  Better performance, scalability and availability translates into cost savings for Web site operators. Because fewer application Web servers are required to meet the challenges posed by traffic spikes and denial of service attacks, Oracle9*i*AS Web Cache offers a simple and inexpensive means of reducing a Web site's cost for each request.

- Network Traffic Reduction

  Most requests are resolved by Oracle9*i*AS Web Cache, reducing traffic to the application Web servers. The cache also reduces traffic to backend databases located on computers other than the application Web server.

Features of Oracle9*i*AS Web Cache

The main features of Oracle9*i*AS Web Cache make it a perfect caching service for e-business Web sites that host online catalogs, news services, and portals. These features include:

- Static and Dynamically Generated Content Caching
- Cache Invalidation
- Performance Assurance
- Site Support
- Cache Hierarchies
- Cache Clusters
- Application Web Server and Proxy Server Features
- Security Features
- Compression

Static and Dynamically Generated Content Caching

Caching rules determine which documents Oracle9*i*AS Web Cache caches. Rules fall into three categories:

- Rules for static content, such as GIF, JPEG, or static HTML files
- Rules for dynamically generated content created using technologies like Java Server Pages (JSP), Active Server Pages (ASP), PL/SQL Server Pages (PSP), Java Servlets, and Common Gateway Interface (CGI). Support of these technologies enables Oracle9*i*AS Web Cache to recognize rules for the following:
  - Multiple-version documents for the same URL, that is, the same URL with slightly different content
  - Session-aware rules for pages containing session information
  - Personalization rules for pages containing personalized greetings, such as "Welcome *Name*," and session-encoded URLs
- Pages that require content assembly of dynamic Edge Side Includes (ESI) fragments

See Also:

- "Caching Dynamically Generated Content" on page 2-11 for further information about dynamically-generated content
- "Content Assembly and Partial Page Caching" on page 2-26

Cache Invalidation

Oracle9*i*AS Web Cache supports **invalidation** as a mechanism to keep the cache consistent with the content on the application Web servers, origin databases, or other dynamically generated means.

Administrators can invalidate cache content in one of two ways:

- Send an invalidation message to the computer running Oracle9*i*AS Web Cache

  When documents are invalidated and a browser requests them, Oracle9*i*AS Web Cache refreshes them with new content from the application Web server.

- Assign an expiration time limit to the documents

  When a document expires, Oracle9*i*AS Web Cache treats it like an invalid document. If the document is requested by a browser, Oracle9*i*AS Web Cache refreshes it with updated content from the application Web server.

> **See Also:** "Cache Freshness and Performance Assurance" on page 2-4 for further information about invalidation

Performance Assurance

When a large number of documents have been invalidated, the retrieval of a new documents can result in overburdened application Web servers.

To handle performance issues while maintaining cache consistency, Oracle9*i*AS Web Cache uses built-in **performance assurance heuristics** that enable it to assign a queue order to documents. These heuristics determine which documents can be served stale and which documents must be refreshed immediately. Documents with a higher priority are refreshed first. Documents with a lower priority are refreshed at a later time.

The queue order of documents is based on the popularity of documents and the validity of documents assigned during invalidation. If the current load and capacity of the application Web server is not exceeded, then the most popular and least valid documents are refreshed first.

> **See Also:** "Cache Freshness and Performance Assurance" on page 2-4 for further information about performance assurance

Site Support

Oracle9*i*AS Web Cache caches and assembles dynamic content for one or more Web sites. From the perspective of Oracle9*i*AS Web Cache, a site can be either a **virtual host site** or an **ESI provider site**. Depending on the site category, you can configure Oracle9*i*AS Web Cache to perform different functions.

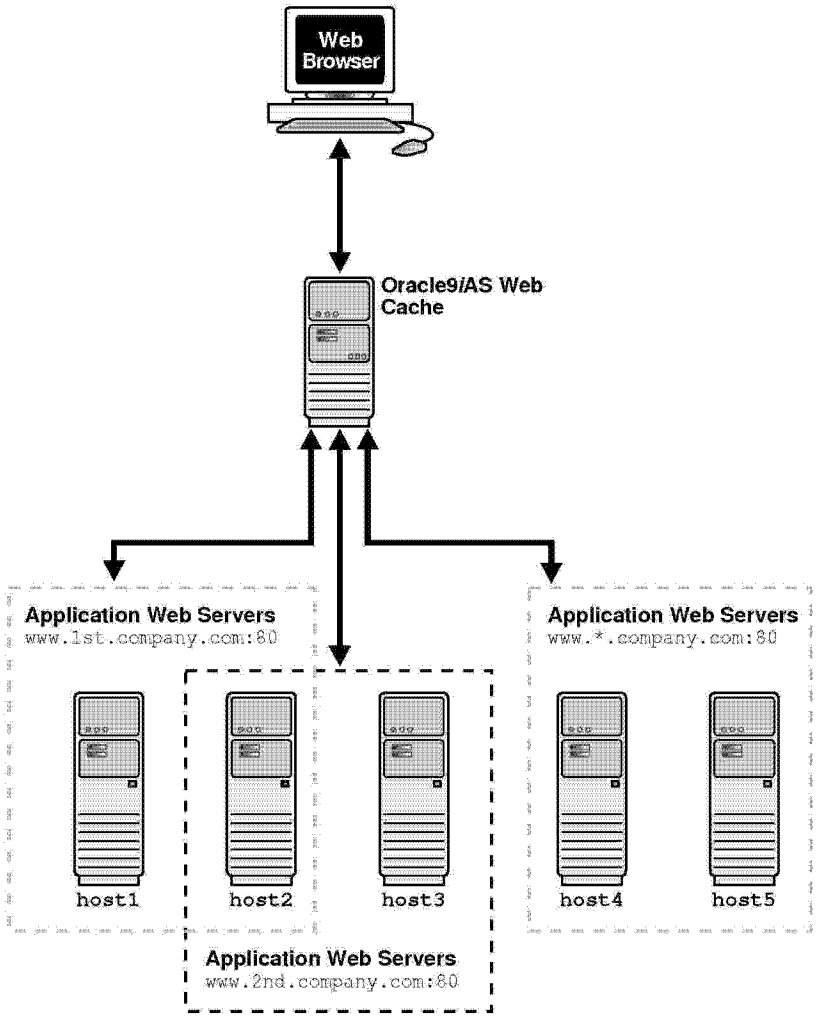
This section covers the following topics:

- Virtual Host Sites
- ESI Provider Sites
- Site Definitions and Site-to-Server Mappings
- How Oracle9iAS Web Cache Locates Application Web Severs or Proxy Servers

Virtual Host Sites

Virtual host sites are sites hosted by Oracle9*i*AS Web Cache. In other words, browsers can request cached content from these sites through Oracle9*i*AS Web Cache. Figure 1–3 on page 1-11 shows Oracle9*i*AS Web Cache caching content for two sites, `www.1st.company.com` and `www.2nd.company.com`. An additional mapping of `www.*.company.com` uses *, enabling additional virtual sites that map to `host4` and `host5` to be added. In addition to caching content, Oracle9*i*AS Web Cache can also assemble ESI fragments from these sites.

Figure 1–3 Multiple Virtual Host Sites



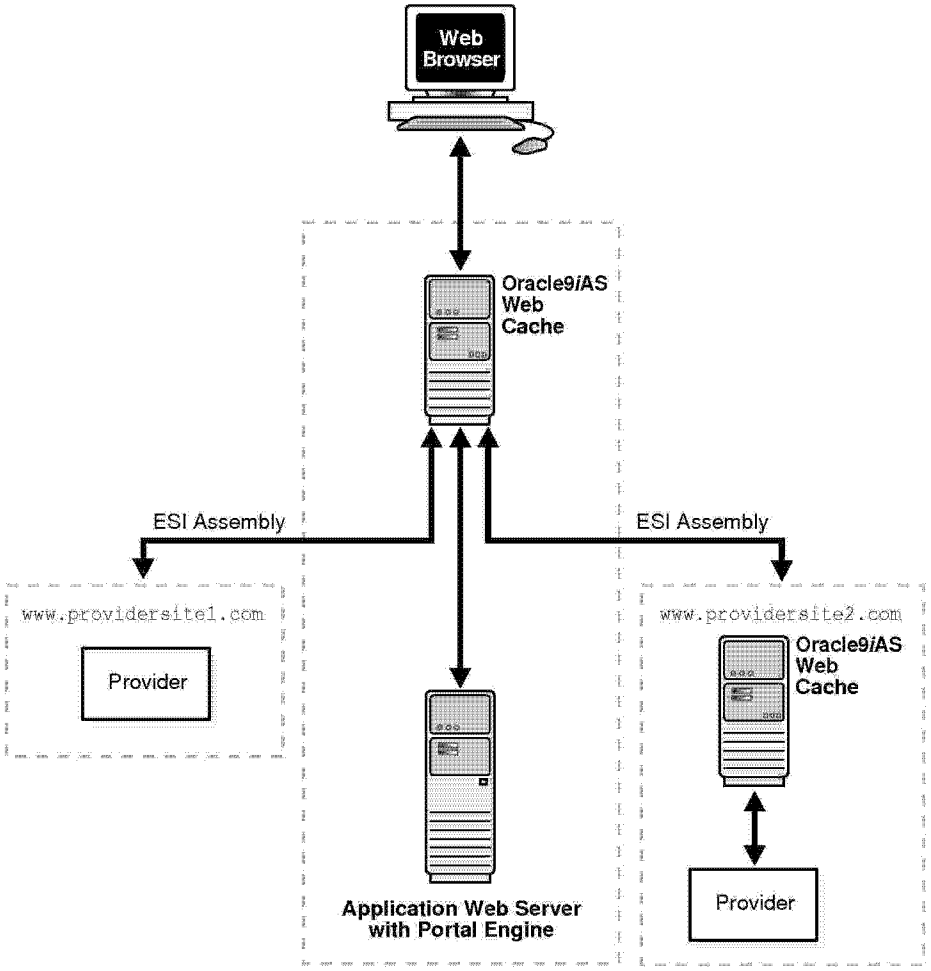
ESI Provider Sites

ESI provider sites are those sites that Oracle9*i*AS Web Cache contacts for ESI assembly only. Browsers are not allowed to request content from these sites.

Figure 1–4 on page 1-13 shows an ESI provider site configuration. In this configuration, Oracle9*i*AS Web Cache receives a request for a page with ESI markup tags. Oracle9*i*AS Web Cache sends the request to the application Web server. The application Web server uses a portal application to create a template page and sends it back to Oracle9*i*AS Web Cache for assembly. Oracle9*i*AS Web Cache includes the ESI fragments for the template page directly from the `www.providersite1.com` site and another Oracle9*i*AS Web Cache server, which is caching content for the `www.providersite2.com` site.

> **See Also:** "Content Assembly and Partial Page Caching" on page 2-26 for further information about ESI

Figure 1–4 Multiple ESI Provider Sites



ORCL00272396

Site Definitions and Site-to-Server Mappings

In order for Oracle9*i*AS Web Cache to recognize a virtual host site or an ESI provider site, administrators need to perform the following:

1. Specify a site definition:
   - Specify the name of the site
   - Specify the listening port number of the site
   - Specify the aliases for the site

   Many sites are represented by one or more aliases. Oracle9*i*AS Web Cache is able to recognize and cache requests for a site and its aliases. For example, site `www.company.com` may have an alias of `company.com`. By specifying this alias, Oracle9*i*AS Web Cache caches the same content from either `company.com` or `www.company.com`.
2. Determine if Oracle9*i*AS Web Cache communicates with application Web servers or proxy servers.

   Oracle9*i*AS Web Cache uses application Web servers for internal sites and proxy servers for external sites protected by a firewall.
3. Specify the host name and listening port number of the application Web server or proxy server.
4. Map the site to the application Web servers or proxy servers.
5. Create caching rules that apply to the site and global rules that apply to all sites.

   The site-specific caching rules are given a higher priority than the global rules.

Note: It may not be possible to specify a site definition for all external ESI provider sites. If an ESI request is made to a provider that does not match any application Web server mapping, then Oracle9*i*AS Web Cache uses Domain Name System (DNS) to resolve the site name. Note that this will not work if there is a firewall between the cache and the ESI provider. In that case, you must provide a proxy server mapping that directs the request to the appropriate proxy.

Undefined ESI provider sites disable the following Oracle9*i*AS Web Cache features:

- Performance assurance heuristics
- Application Web server and proxy server features, such as surge protection, load balancing, failover, and session binding

---

To further understand the mappings, reconsider Figure 1–3 on page 1-11. Web sites `www.1st.company.com:80` and `www.2nd.company.com:80` can have site aliases of `1st.company.com:80` and `2nd.company.com:80`, respectively. The site to application Web server mappings are as follows:

- `www.1st.company.com` maps to application Web servers `host1` and `host2`
- `www.2nd.company.com` maps to application Web servers `host2` and `host3`
- `www.*.company.com` maps to `host4` and `host5`

How Oracle9*i*AS Web Cache Locates Application Web Severs or Proxy Servers

When Oracle9*i*AS Web Cache receives a browser request for a document, it determines the destination site using one of the following elements:

- `Host` request-header field from the request
- Host portion of the requested URL
- `src` attribute of the ESI `<esi:include>` tag

Oracle9*i*AS Web Cache then looks up the configured site settings and mappings to determine if the site is supported, and the application Web servers or proxy servers and caching rules for the site. If there are no site settings and mappings for external ESI provider sites, Oracle9*i*AS Web Cache uses Domain Name System (DNS) to resolve the site name.

If the request does not include host information, then Oracle9*i*AS Web Cache sends the request to the default site. A default site is established for the Oracle HTTP Server when Oracle9*i* Application Web Server is installed. You can specify another site to be the default site.

See Also:

- "Using Oracle9iAS Web Cache to Support Multiple Sites"for deployment scenarios with application Web servers and proxy servers
- Chapter 6, "Initial Setup and Configuration" for configuration details
- "Default Site Settings" on page 6-32 for default site settings

Cache Hierarchies

For high availability and performance, many Internet businesses mirror their Web sites in strategic geographical locations. You can deploy Oracle9*i*AS Web Caches in a **cache hierarchy** so that an Oracle9*i*AS Web Cache server caches content from another Oracle9*i*AS Web Cache to a local market. Caches serving local markets shortens response time to these markets and reduces bandwidth and rack space costs for the content provider.

Oracle9*i*AS Web Cache provides supports two kinds of cache hierarchies:

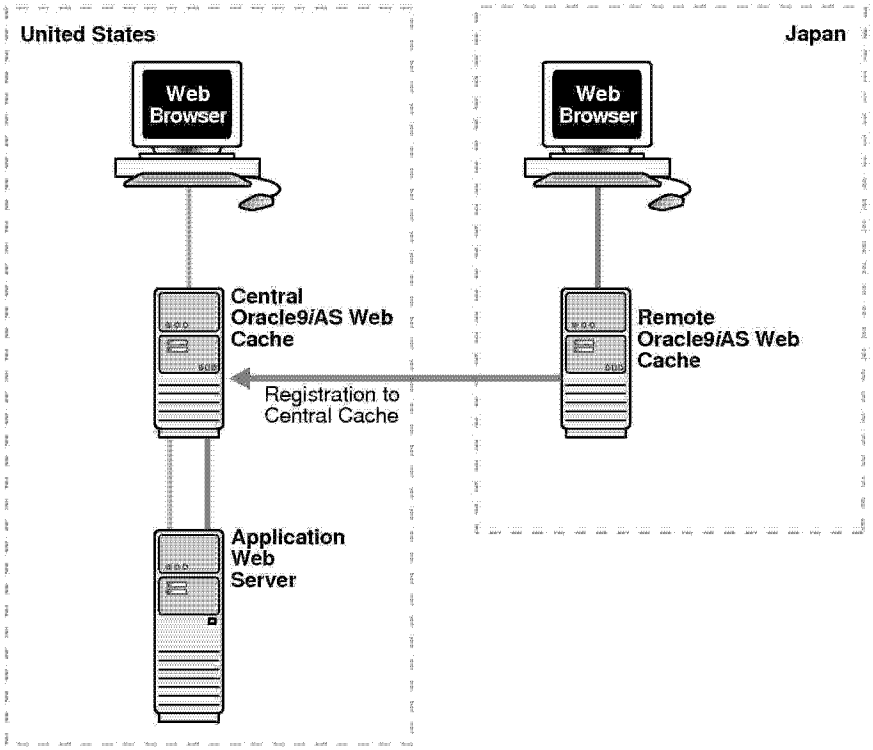
- A **distributed cache hierarchy** in which a **central cache** acts as an application Web server to a **remote cache**.
- An **ESI cache hierarchy** in which a **provider cache** acts as an application Web server to a **subscriber cache**.

A distributed cache hierarchy centralizes the management of application logic and data to the central cache and provides remote assembly and delivery of content. Compared with full-scale mirroring and database replication, a distributed cache hierarchy provides a more cost-effective model of distributed computing.

The remote cache is configured with the central cache as its application Web server. When the remote cache requests content from the central cache to serve a request, the remote cache identifies itself as an Oracle9*i*AS Web Cache to the central cache during a transparent registration process. Once registration is complete, the central cache establishes a hierarchical relationship with the remote cache. Registration also enables invalidation messages to be propagated from the central cache to the remote cache.

Figure 1–5 depicts a distributed cache hierarchy. A central cache resides in the United States office and a remote cache resides in the Japan. While the central cache in United States caches content from an application Web server, the remote cache in Japan caches content from the central cache.

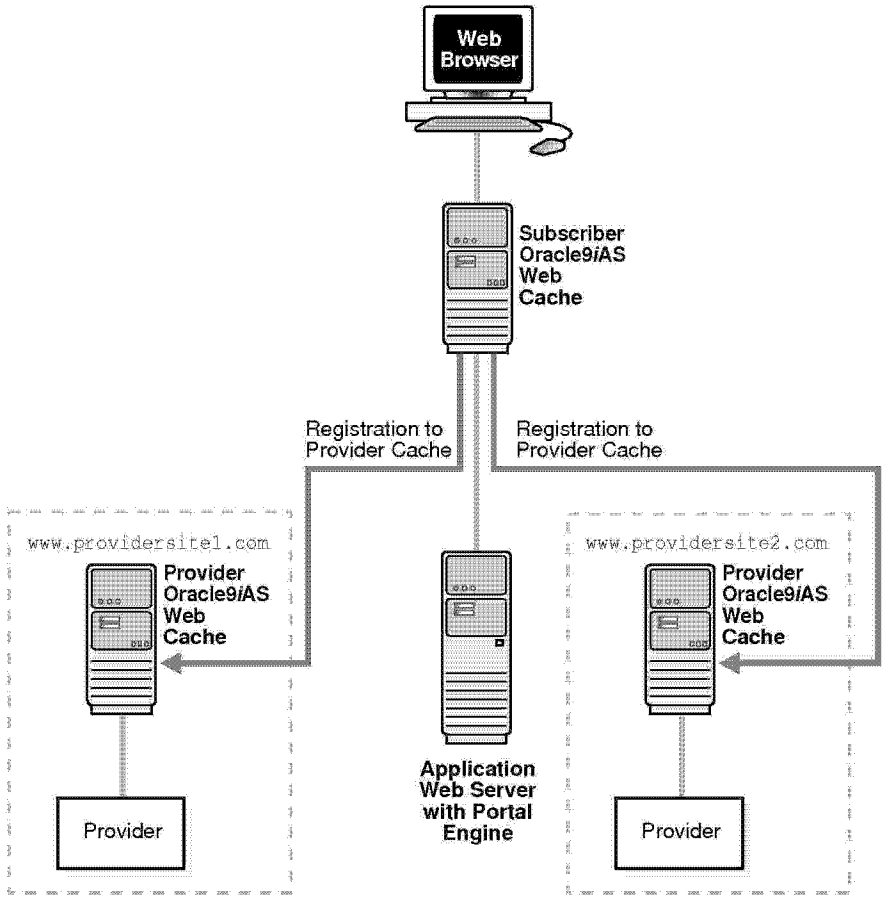
Figure 1–5 Distributed Cache Hierarchy



In an ESI cache hierarchy, the subscriber cache is configured with the provider caches as its application Web servers. When the subscriber cache requests content from the provider caches for ESI assembly, the subscriber cache identifies itself as an Oracle9*i*AS Web Cache to the provider caches during a transparent registration process. Once registration is complete, the provider caches establish a hierarchical relationship with the subscriber cache. Registration also enables invalidation messages to be propagated from the provider caches to the subscriber cache.

Figure 1–6 depicts an ESI cache hierarchy. A subscriber cache performs ESI assembly. Provider caches locally cache ESI fragments for ESI provider sites `www.providersite1.com` and `www.providersite2.com`. During ESI page assembly, the subscriber cache contacts the provider caches for the ESI fragments. By caching the ESI fragments locally on the provider caches, fragments are cached both by the provider and subscriber caches. This provides for quick page assembly.

Figure 1–6 ESI Cache Hierarchy



See Also:

- "Invalidation Propagation" on page 2-7 for information about how invalidation messages are propagated in a hierarchy of Oracle9*i*AS Web Cache servers
- "Configuring a Hierarchy of Caches" on page 6-47 for configuration details

Cache Clusters

To increase the availability and scalability of your Web site, you can configure multiple instances of Oracle9*i*AS Web Cache to run as members of a cache cluster. A cache cluster is a loosely coupled collection of cooperating Oracle9*i*AS Web Cache 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 Web caches, more content can be cached and more client connections can be supported, expanding the capacity of your Web site.

See Also:

- Chapter 3, "Cache Clustering"for more information about cache clusters
- "Configuring a Cache Cluster" on page 6-53 for configuration details

Application Web Server and Proxy Server Features

Oracle9*i*AS Web Cache provides the following features for the application Web server and proxy server it supports:

- Surge Protection
- Load Balancing
- Backend Failover
- Session Binding

Where applicable, the term origin server is used in place of application Web server or proxy server to simplify the concepts presented in this section.

Surge Protection

Oracle9*i*AS Web Cache passes requests for non-cacheable or stale documents to the origin servers. To prevent an overload of requests on the origin servers, Oracle9*i*AS Web Cache has a surge protection feature that enables you to set a limit on the number of concurrent requests that the origin servers can handle. When the limit is reached, subsequent requests are queued to wait up to a maximum amount of time. If the maximum wait time is exceeded, Oracle9*i*AS Web Cache rejects the request and serves a site busy apology page to the Web browser that initiated the request.

Load Balancing

Most Web sites are served by multiple origin servers running on multiple computers that share the load of HTTP and HTTPS requests. All requests that Oracle9*i*AS Web Cache cannot serve are passed to the origin servers. Oracle9*i*AS Web Cache balances the load among origin servers by determining the percentage of the available capacity, the **weighted availability capacity** of each origin server. Oracle9*i*AS Web Cache sends a request to the origin server with the highest weighted available capacity. The weighted available capacity is determined by the following formula:

```
(Capacity - Load) / Capacity
```

where:

- `Capacity` is the maximum number of concurrent connections that the origin server can accept
- `Load` is the number of connections currently in use

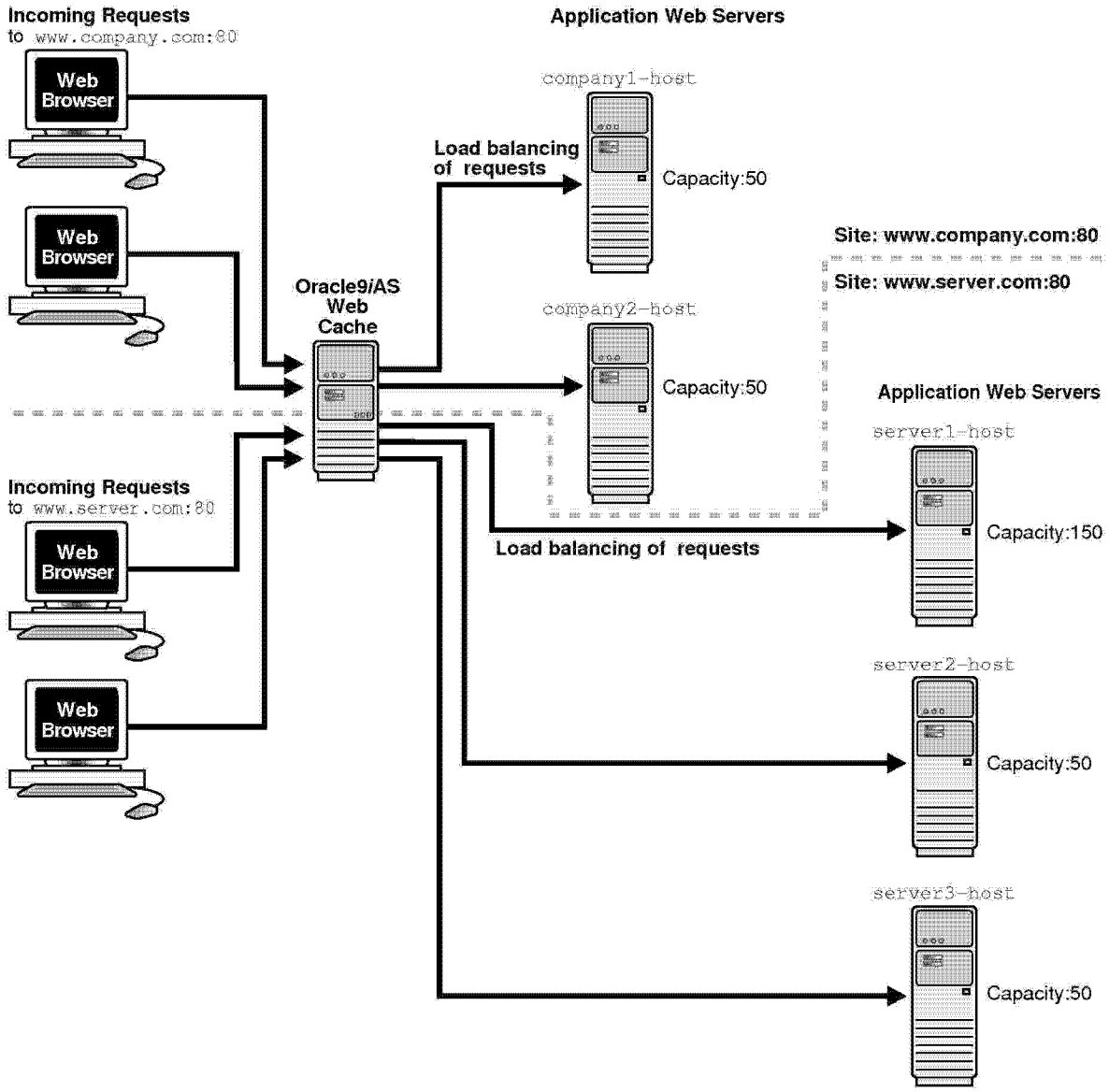
If the weighted available capacity is equal for multiple origin servers, then Oracle9*i*AS Web Cache sends requests to the origin servers in round-robin fashion. With round-robin, the first origin server in the list of configured servers receives the request, then the second origin server receives the second request.

If the weighted available capacity is not equal, Oracle9*i*AS Web Cache sends the request to the origin server with the highest weighted available capacity.

To configure load balancing, set the capacity of each origin server.

Figure 1–7 on page 1-22 shows two sites, `www.company.com:80` and `www.server.com:80`. `www.company.com:80` is supported by application Web servers `company1-host` and `company2-host` with capacities of 50 each. `www.server.com:80` is supported by application Web servers `server1-host`, `server2-host`, and `server3-host` with capacities of 150, 50, and 50, respectively.

Figure 1–7 Load Balancing



Assuming all application Web servers have an initial load of 0, the requests to `www.company.com:80` and `www.server.com:80` will be distributed in the following manner:

- The requests to `company1-host` and `company2-host` will be distributed between the two origin servers so that they maintain an equal load. The first request is sent to `company1-host`. The second request is sent to `company2-host` if `company1-host` is still processing the first request. The third and subsequent requests are sent to the origin server that has the highest weighted available capacity.
- The first request to `www.server.com:80` is sent to `server3-host`. The second request is sent to `server2-host`, because `server1-host` now has a weighted available capacity of 99.3 percent and `server2-host` has a weighted available capacity of 100 percent. The third request is sent to `server3-host` because `server2-host` now has a weighted available capacity of 98 percent and `server3-host` has a weighted available capacity of 100 percent. The fourth request is sent to `server1-host` because `server2-host` and `server3-host` now have weighted available capacities of 98 percent. The fifth request is sent to `server1-host` because its weighted available capacity is 98.6 percent, still greater than that of `server2-host` and `server3-host`.

See Also:

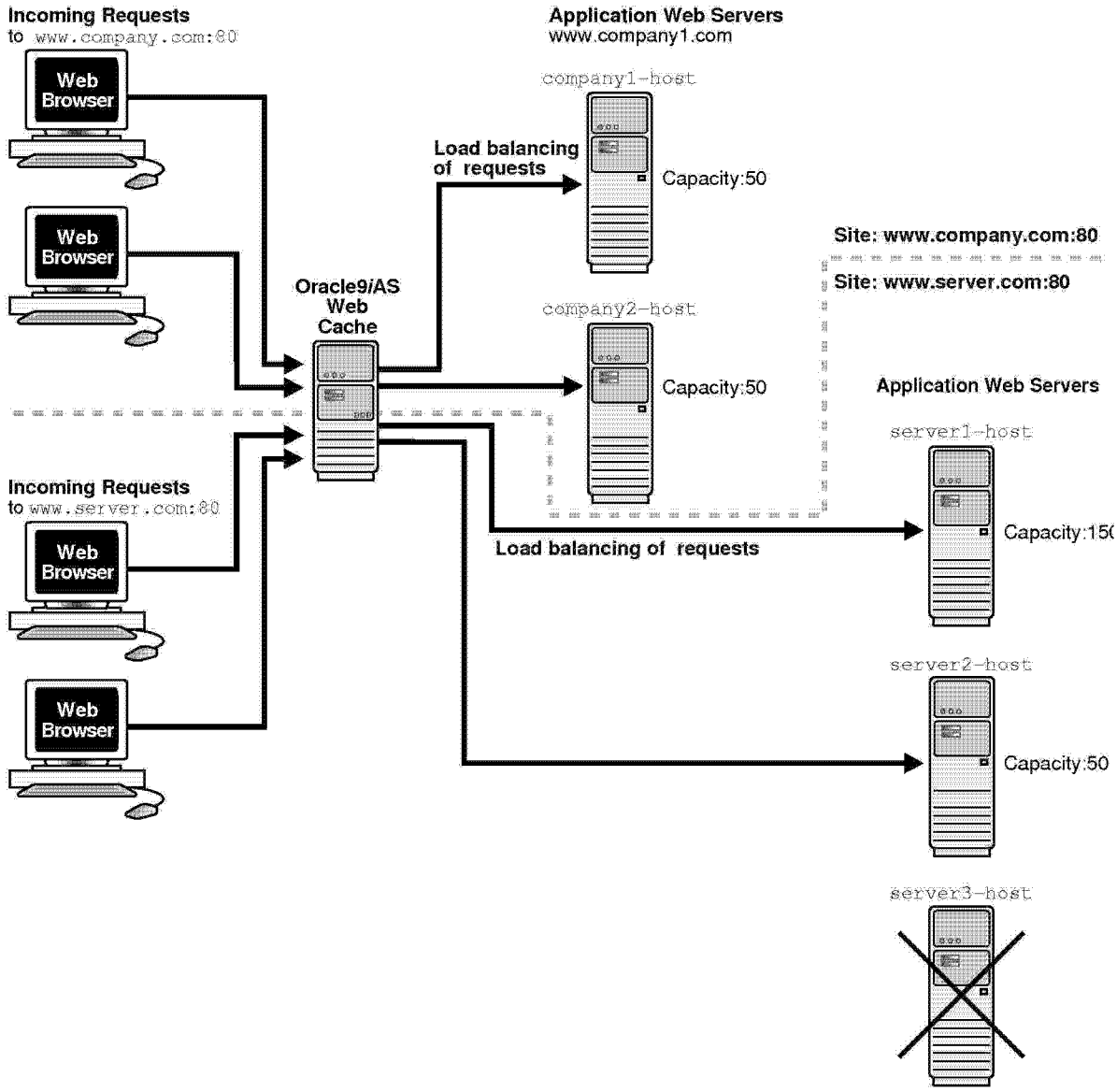
- "Task 9: Specify Settings for Origin Servers" on page 6-23 for application Web server or proxy server configuration
- "Configuring Load Balancing and Failover" on page 6-43 for load balancing and failover configuration

Backend Failover

After a specified number of continuous request failures, Oracle9*i*AS Web Cache considers an origin server as failed. When an origin server fails, Oracle9*i*AS Web Cache automatically distributes the load over the remaining origin servers based on the available load. Oracle9*i*AS Web Cache polls the failed origin server for its current up/down status until it is back online. When the failed server returns to operation, Oracle9*i*AS Web Cache will include its weighted available capacity to load balance requests.

The failover feature is shown in Figure 1–8 on page 1-24. An outage of `server3-host`, which had a capacity of 50, results in 75 percent of requests being distributed to `server1-host` and 25 percent request being distributed to `server2-host`.

Figure 1–8 Failover



Session Binding

Oracle9*i*AS Web Cache supports Web sites that use a session ID or **session cookie** to bind user sessions to a given origin server in order to maintain state for a period of time. To utilize the **session binding** feature, the origin server itself must maintain state, that is, it must be stateful. Web sites bind user sessions by including session data in the HTTP header or body it sends to Web browsers in such a way that the browser is forced to include it with its next request. This data is transferred either with parameters embedded in the URL or cookies, which are text strings stored on the client.

Figure 1–9 on page 1-26 shows how Oracle9*i*AS Web Cache supports documents that use session binding:

1. When a request first comes in, Oracle9*i*AS Web Cache uses load balancing to determine to which origin server the request is forwarded. In this example, application Web server `www.server2.com` is selected.
2. If the requested document requires session binding, the origin server sends the session information back to the browser through Oracle9*i*AS Web Cache in the form of a cookie or an **embedded URL parameter**.
3. Oracle9*i*AS Web Cache sends subsequent requests for the session to the origin server that established the session, bypassing load balancing. In this example, application Web server `www.server2.com` handles the subsequent requests.

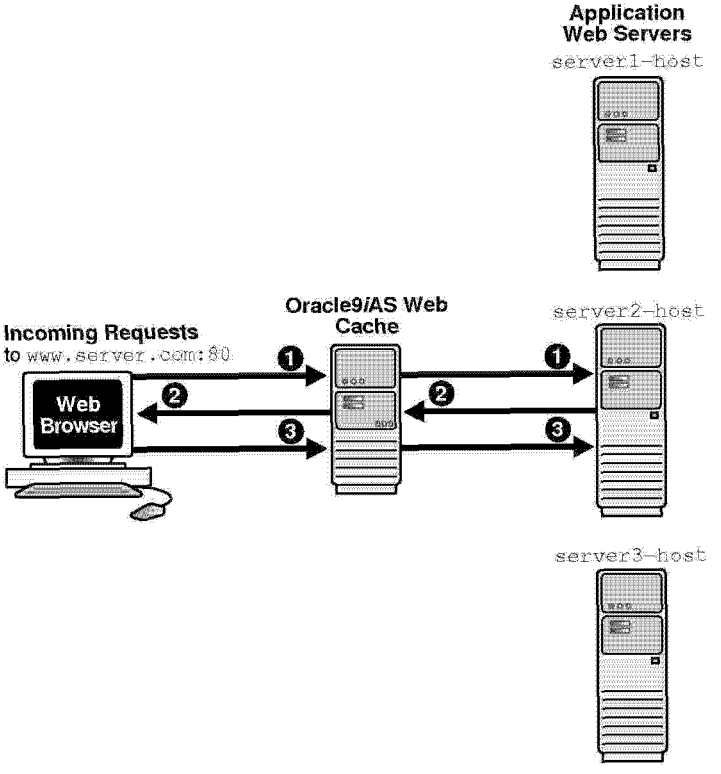
To configure session binding, specify a session definition that specifies the name of session cookie or embedded URL parameter.

> **See Also:** "Binding a Session to an Origin Server" on page 6-44 for configuration details

---

Note: If an origin server is busy, then Oracle9*i*AS Web Cache disables session binding to that origin server.

Figure 1–9 Session Binding



ORCL00272409

Security Features

Oracle9*i*AS Web Cache provides the following security-related features:

- Restricted Administration
- Secure Sockets Layer (SSL) Support

> **Note:** Oracle9*i*AS Web Cache does not cache pages that support basic HTTP authentication. These pages result in cache misses.

Restricted Administration

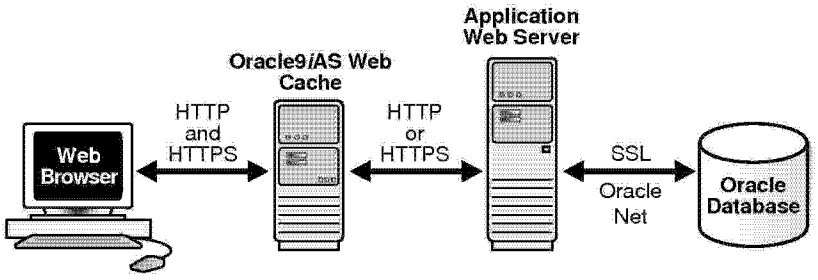
Oracle9*i*AS Web Cache restricts administration with the following features:

- Password authentication for administration and invalidation operations
- Control over which ports administration and invalidation operations can be requested from
- IP and subnet administration restrictions

Secure Sockets Layer (SSL) Support

The Secure Sockets Layer (SSL) protocol, developed by Netscape Corporation, is an industry-accepted standard for network transport layer security. SSL provides authentication, encryption, and data integrity, in a public key infrastructure (PKI). By supporting SSL, Oracle9*i*AS Web Cache is able to cache pages for HTTPS requests. As shown in Figure 1–10, you can configure Oracle9*i*AS Web Cache to receive HTTPS browser requests and send HTTPS requests to the origin servers.

Figure 1–10 SSL for Secure Connections



ORCL00272410

When sending requests to origin servers, note that HTTPS traffic can be processor intensive. If Oracle9*i*AS Web Cache needs to have traffic travel over the open Internet, then configure Oracle9*i*AS Web Cache to send HTTPS requests to the origin servers. If traffic only travels through a LAN in a data center, then the traffic can be sent with HTTP so as to reduce the load on the origin servers.

Limitations: HTTPS support has the following limitations in this release:

- Oracle9*i*AS Web Cache does not provide authentication or access control
- Oracle9*i*AS Web Cache does not support client-side certification

SSL interacts with the following entities:

- Certificate Authority
- Certificate
- Wallet

Certificate Authority A certificate authority (CA) is a trusted third party that certifies the identity of third parties and other entities, such as users, databases, administrators, clients, and servers. The certificate authority verifies the party identity and grants a certificate, signing it with its private key. The Oracle9*i*AS Web Cache certificate must be signed by a CA.

Different CAs may have different identification requirements when issuing certificates. One may require the presentation of a user's driver's license, while another may require notarization of the certificate request form, or fingerprints of the requesting party.

The CA publishes its own certificate, which includes its public key. Each network entity has a list of certificates of the CAs it trusts. Before communicating with another entity, a given entity uses this list to verify that the signature on the other entity's certificate is from a known, trusted CA.

Network entities can obtain their certificates from the same or different CAs. By default, Oracle Advanced Security automatically installs trusted certificates from VeriSign, RSA, Entrust, and GTE CyberTrust when you install a new wallet

Certificate A certificate is created when a party's public key is signed by a trusted CA. A certificate ensures that a party's identification information is correct, and that the public key actually belongs to that party.

A certificate contains the party's name, public key, and an expiration date—as well as a serial number and certificate chain information. It can also contain information about the privileges associated with the certificate.

When a network entity receives a certificate, it verifies that it is a trusted certificate—one issued and signed by a trusted certificate authority. A certificate remains valid until it expires or is terminated.

Wallet A wallet is a transparent database used to manage authentication data such as keys, certificates, and trusted certificates needed by SSL. A wallet has an X.509 version 3 certificate, private key, and list of trusted certificates.

Security administrators use the Oracle Wallet Manager to manage security credentials on the Oracle9*i*AS Web Cache server. Wallet owners use it to manage security credentials on clients. Specifically, Oracle Wallet Manager is used to do the following:

- Generate a public-private key pair and create a certificate request for submission to a certificate authority.
- Install a certificate for the identity.
- Configure trusted certificates for the identity.

To support HTTPS for Oracle9*i*AS Web Cache, create a wallet on the Oracle9*i*AS Web Cache server for each supported site. When creating listening ports for Oracle9*i*AS Web Cache, specify the location of the wallet. One wallet can be shared among all the listening ports, or a separate wallet can be created for each port.

See Also:

- "Configuring Oracle9iAS Web Cache for HTTPS Requests" on page 6-37 for configuration details
- *Oracle9i Application Server Security Guide* for further information about Oracle Wallet Manager

How SSL Works To describe how SSL works in an HTTPS connection, the word client is used to describe either a browser or Oracle9*i*AS Web Cache, and the word server is used to describe either Oracle9*i*AS Web Cache or an origin server.

The authentication process between the client and server consists of the steps that follow:

1. The client initiates a connection to the server by using HTTPS.
2. SSL performs the handshake between the client and the server.

At the commencement of an HTTPS network connection between the client and server, an SSL handshake is performed. An SSL handshake includes the following actions:

- The client and server establish which cipher suites to use.
- The server sends its certificate to the client, and the client verifies that the server's certificate was signed by a trusted CA.
- The client and server exchange key information using public key cryptography; based on this information, each generates a session key. All subsequent communications between the client and the server is encrypted and decrypted by using this set of session keys and the negotiated cipher suite.

Compression

You can select to have Oracle9*i*AS Web Cache compress both cacheable and non-cacheable documents upon insertion into the cache for browsers. Because compressed documents are smaller in size, they are delivered faster to browsers with fewer round-trips, reducing overall latency. On average, Oracle9*i*AS Web Cache is able to compress text files by a factor of 4. For example, 300 KB files are compressed down to 75 KB.

Compatibility with Oracle9*i*AS Components

Table 1–1 describes Oracle9*i*AS Web Cache compatibility with other Oracle9*i*AS components.

Table 1–1 Oracle9iAS Web Cache Compatibility with Oracle9iAS Components

Oracle9*i*AS Component	Description
Oracle9*i*AS Clickstream Intelligence	Oracle9*i*AS Discoverer access logs import easily into Oracle9*i*AS Clickstream Intelligence, which provides a rich set of data warehousing and clickstream analysis functionality. **See Also:** "Configuring Access Logs" on page 8-59 and *Oracle9iAS Clickstream Intelligence Administrator's Guide*
Oracle9*i*AS Discoverer	Starting with Oracle9*i* Application Server release 2, Oracle9*i*AS Discoverer has been closely integrated with Oracle9*i*AS Web Cache to improve Discoverer Viewer's overall scalability, performance, and availability. Oracle9*i*AS Web Cache ships with a number of predefined caching rules for this purpose, and Oracle9*i*AS Discoverer uses ESI `Surrogate-Control` headers to govern cacheability of other non-configured responses. Because of this integration, the load on mid-tier and database servers in Oracle9*i*AS Discoverer deployments is reduced, more Discoverer Viewer users are able to access the system concurrently, and those users experience significantly better response times for workbook operations and common business intelligence queries. **See Also:** *Oracle9iAS Discoverer Configuration Guide*
Oracle9*i*AS Forms Services	Oracle9*i*AS Web Cache does not currently work applications that use Oracle9*i*AS Forms Services.
Oracle9*i*AS Portal	Oracle9*i*AS Web Cache has been closely integrated with Oracle9*i*AS Portal to improve Portal's overall scalability, performance and availability. Oracle9*i*AS Portal ships with a number of pre-defined caching and invalidation policies that ensure optimal use of Oracle9*i*AS Web Cache. Oracle9*i*AS Web Cache controls have been built into the Oracle9*i*AS Portal administrative user interface and can also be specified by content providers through the Portlet Developer Kit (PDK). **See Also:** Oracle9*iAS Portal* online help and *Oracle9iAS Portal Configuration Guide*
Oracle9*i*AS Reports Services	Oracle9*i*AS Web Cache cannot be used to accelerate Oracle9*i*AS Reports Services in this release.

Table 1–1 (Cont.) Oracle9iAS Web Cache Compatibility with Oracle9iAS Components

Oracle9*i*AS Component	Description
Oracle9*i*AS Single Sign-On	Applications that use Oracle9*i*AS Single Sign-On can take advantage of Oracle9*i*AS Wireless. Both Oracle9*i*AS Web Cache and `mod_sso`, a feature of Oracle HTTP Server, have been configured out of the box to ensure that single sign-on authentication requests are tunneled transparently through Oracle9*i*AS Web Cache. No additional configuration is required on the customer side. **See Also:** *Oracle9iAS Single Sign-On Administrator's Guide*
Oracle9*i*AS Wireless	Oracle9*i*AS Wireless is integrated with Oracle9*i*AS Web Cache to improve page rendering performance and scalability. It should be noted that Oracle9*i*AS Web Cache does not understand WAP and is not used by Oracle9*i*AS Wireless in the traditional sense in that the cache does not "front-end" the wireless server. Instead, the cache is used as a repository for post-transformed content; the wireless runtime determines what content needs to be inserted into the cache and when to expire content in the cache. Oracle9*i*AS Wireless, in this case, acts as a device adaptation cache rather than a reverse-proxy cache. Since markup content is cached using Oracle9*i*AS Wireless, the performance and scalability benefits are due to two factors—reduced device adaptation costs and significantly reduced adapter invocation costs. The savings in terms of device adaptation costs stem form the fact that content that can be shared across users and sessions is essentially transformed only once (for each logical device) from its Mobile XML format. Secondly, since the content is not generated every time by an adapter, the total adapter invocation cost is significantly educed for a site that has a large subset of cacheable pages. **See Also:** *Oracle9iAS Wireless Getting Started and System Guide*

Exhibit 23

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 24

THIS EXHIBIT HAS BEEN REDACTED IN ITS ENTIRETY

Exhibit 25

Oracle® Application Server

Best Practices Guide

10*g* Release 2 (10.1.2)

B28654-01

March 2006

ORACLE

Oracle Application Server Best Practices Guide, 10*g* Release 2 (10.1.2)

B28654-01

Copyright © 2004, 2006, Oracle. All rights reserved.

Primary Author: William Bathurst, Jan Carlin, Fermin Castro, Ron Caneel, Douglas Clark, Mike Donohue, Shalendra Goel, Nicole Haba, Christian Hauser, Pavana Jain, Michael Lehman, John Lang, Sandhya Rajput, Gurudatt Shashikumar, Deborah Steiner, Olaf Stullich, Deepak Thomas, Jinyu Wang, Philip Weckerle

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 USA, Inc., 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, JD Edwards, PeopleSoft, and Siebel are registered trademarks 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.

7

OracleAS Web Cache

This chapter describes performance best practices for OracleAS Web Cache. It includes the following topics:

- Section 7.1, "Improve Performance, Scalability, and Availability"
- Section 7.2, "Planning and Deployment"
- Section 7.3, "Secure Content to Prevent Tampering"
- Section 7.4, "Configuring OracleAS Web Cache"
- Section 7.5, "Increasing Cache Hits"
- Section 7.6, "Invalidation and Expiration"
- Section 7.7, "Optimizing Response Times"

7.1 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 7.5.4, "Use Partial Page Caching Where Possible to Increase Cacheability" on page 7-9 for more information.
- OracleAS Web Cache includes 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 7.2.2, "Cluster Cache Instances to Make Availability, Scalability, and Performance Gains" on page 7-3 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. See Section 7.2.4, "Use OracleAS Web Cache Built-In Load Balancing to Improve Availability and Scalability of Origin Servers" on page 7-4 for more information.
- OracleAS Web Cache is designed to run with commodity hardware, reducing the cost. See Section 7.2.1, "Use Two CPUs and Consider Deploying on Dedicated

Hardware to Avoid Operating System Limitations" on page 7-2 for more information.

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.

7.2 Planning and Deployment

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

- Section 7.2.1, "Use Two CPUs and Consider Deploying on Dedicated Hardware to Avoid Operating System Limitations"
- Section 7.2.2, "Cluster Cache Instances to Make Availability, Scalability, and Performance Gains"
- Section 7.2.3, "Use a Hardware Load Balancer in Front of OracleAS Web Caches to Avoid a Single Point of Failure"
- Section 7.2.4, "Use OracleAS Web Cache Built-In Load Balancing to Improve Availability and Scalability of Origin Servers"
- Section 7.2.5, "Test Application Upgrades and Patches to Ensure Existing Cache and Session Rules Still Function Correctly"

7.2.1 Use Two CPUs and Consider Deploying on Dedicated Hardware to Avoid Operating System Limitations

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 for a dual CPU. Because OracleAS Web Cache is an in-memory cache, it is rarely limited by CPU cycles. Additional CPUs do not increase performance significantly. 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.

Implementation Details

See Also: Chapter 7, "Optimizing OracleAS Web Cache," in the *Oracle Application Server Performance Guide*

ORCL00528166

7.2.2 Cluster Cache Instances to Make Availability, Scalability, and Performance Gains

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, you can cache more content and support more client connections, expanding the overall capacity of your Web site and improving its performance.

Implementation Details

See Also: Chapter 3, "Cache Clustering," in the *Oracle Application Server Web Cache Administrator's Guide*

7.2.3 Use a Hardware Load Balancer in Front of OracleAS Web Caches to Avoid a Single Point of Failure

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 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 7.2.4, "Use OracleAS Web Cache Built-In Load Balancing to Improve Availability and Scalability of Origin Servers" on page 7-4.

If you need a low-cost solution to a hardware load balancer and do not require caching support, you can configure OracleAS Web Cache solely as a software load balancer. This configuration mode is useful for managing traffic to a low-volume, departmental, or test Web site.

Implementation Details

See Also:

- Chapter 5, "OracleAS Web Cache Topologies," in the *Oracle Application Server Web Cache Administrator's Guide* for more information about deploying hardware load balancers with OracleAS Web Cache
- Section "OracleAS Web Cache Solely as a Software Load Balancer or Reverse Proxy" in the *Oracle Application Server Web Cache Administrator's Guide* for more information about configuring OracleAS Web Cache as a load balancer

ORCL00528167

7.2.4 Use OracleAS Web Cache Built-In Load Balancing to Improve 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. You can configure OracleAS Web Cache 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, you can deploy two or more nodes running OracleAS Web Cache behind a third-party hardware load-balancing device. Oracle 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 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.

See Also:

- Chapter 1, "Introduction to OracleAS Web Cache," in the *Oracle Application Server Web Cache Administrator's Guide* for more information on load balancing, performance assurance and surge protection functionality
- Whitepapers available from the Oracle Technology Network at `http://www.oracle.com/technology/products/ias/web_cache/index.html`

7.2.5 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,

ORCL00528168

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.

7.3 Secure Content to Prevent Tampering

Depending on the application, you may or may not want requests for secure pages to go through the cache. Be cautious about caching secure content in OracleAS Web Cache. For secure content you do cache, ensure users cannot do URL tampering to see other user content. For this content, 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 or decryption processing time by OracleAS Web Cache.

See Also:

- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server Enterprise Deployment Guide*

7.4 Configuring OracleAS Web Cache

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

- Section 7.4.1, "Configure Enough Memory to Avoid Swapping Objects In and Out of the Cache"
- Section 7.4.2, "Allocate Sufficient Network Bandwidth to Accommodate the Throughput Load"
- Section 7.4.3, "Set a Reasonable Number of Network Connections to Maximize Performance"
- Section 7.4.4, "Create Custom Error Pages to Suit Your Environment"

7.4.1 Configure Enough Memory to Avoid Swapping Objects In and Out of the Cache

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 500 MB.

The amount of memory that OracleAS Web Cache uses to store an object depends on whether the object is larger or smaller than 2 KB:

Implementation Details

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

1. Determine which 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

ORCL00528169

larger than 2 KB. Determine the expected peak load—the maximum number of objects to be processed concurrently.

2. Calculate the amount of memory needed.

> **See Also:** Chapter 8, "Setup and Configuration," in 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

7.4.2 Allocate Sufficient Network Bandwidth to Accommodate the Throughput Load

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 for each 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.

7.4.3 Set a Reasonable Number of Network Connections to Maximize Performance

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, you can affect performance, 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.

Implementation Details

> **See Also:** Chapter 8, "Setup and Configuration," in the *Oracle Application Server Web Cache Administrator's Guide* for information about setting the number of network connections

7.4.4 Create Custom Error Pages to Suit Your Environment

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.

ORCL00528170

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

Implementation Details

> **See Also:** Chapter 8, "Setup and Configuration," in the *Oracle Application Server Web Cache Administrator's Guide* for information on creating or modifying default error pages

7.5 Increasing Cache Hits

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

- Section 7.5.1, "Use Cookies and URL Parameters to Increase Cache-hit Ratios"
- Section 7.5.2, "Use Redirection to Cache Entry Pages"
- Section 7.5.3, "Use Surrogate-Control Headers Instead of Caching Rules to Better Manage Cacheability"
- Section 7.5.4, "Use Partial Page Caching Where Possible to Increase Cacheability"
- Section 7.5.5, "Use ESI Variables to Improve Cache-hit Ratios for Personalized Pages"
- Section 7.5.6, "Use the <esi:environment> Tag to Authenticate or Authorize Callbacks"
- Section 7.5.7, "Use JESI to Cache JSP Output"

7.5.1 Use Cookies and URL Parameters to Increase Cache-hit Ratios

OracleAS Web Cache can cache different versions of an object 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 objects 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 objects, so a single object 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 object with placeholders for the embedded URL parameters or the personalized strings, and dynamically substitute the correct values into the placeholders when serving the object to clients.

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

ORCL00528171

Implementation Details

See Also: Chapter 2, "Caching Concepts," in the *Oracle Application Server Web Cache Administrator's Guide* for more information on multiple-version objects, sessions, ignoring URL parameter values, simple personalization, and how to control whether OracleAS Web Cache serves a cached object based on sessions

7.5.2 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.

Implementation Details

See Also: Chapter 12, "Creating Caching Rules," in the *Oracle Application Server Web Cache Administrator's Guide* for more information on configuring caching rules for pages requiring session establishment,

7.5.3 Use Surrogate-Control Headers Instead of Caching Rules to Better Manage Cacheability

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 Application Server Control Console or OracleAS Web Cache Manager interface.
- Application developers can set caching policies through 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 an object, it is preferable to use the `Surrogate-Control` header.

If you define many different categories of cacheable and non-cacheable objects, you need to carefully define rule selectors and rule priorities so that the appropriate rule is used for any object. 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 objects that OracleAS Web Cache can receive from an origin server, then editing the

ORCL00528172

configuration using one of the administration interfaces may be simpler than generating `Surrogate-Control` headers for many objects.

Often, a combination of the two approaches is best.

Implementation Details

> **See Also:** Chapter 12, "Creating Caching Rules," in the *Oracle Application Server Web Cache Administrator's Guide* for more information about the `Surrogate-Control` response header

7.5.4 Use Partial Page Caching Where Possible to Increase Cacheability

Many Web pages, such as portal pages, are composed of fragments with unique caching properties. For these pages, full-page caching is not feasible. 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. Different templates can share the same fragment, 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 is 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. You can cache each fragment in a full Web page 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, you can update particular fragments rather than 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.

7.5.5 Use ESI Variables to Improve Cache-hit Ratios 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 OracleAS Web Cache can resolve variables to different pieces of request information or response information, you can significantly reduce the uniqueness of templates and fragments 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.

ORCL00528173

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. Response variable occurrences in the enclosing template can access an **ESI environment**, which is a type of fragment with a response that defines a set of variables. 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 appointment 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 operation reduces storage and retrieval overhead similarly.

7.5.6 Use the <esi:environment> Tag to Authenticate or Authorize Callbacks

Some applications protect certain Web pages with authentication, authorization, or session validation in the HTTP request. Even though you can cache page content, 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 more than 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 terminate this page or output an alternative error page or login page.

Implementation Details

See Also: Chapter 16, "Edge Side Includes (ESI) Language Tags," in the *Oracle Application Server Web Cache Administrator's Guide* for more information about using the `<esi:environment>` tag or implementing ESI exception handling

7.5.7 Use JESI to Cache JSP Output

In dynamic applications, you can use 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

ORCL00528174

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.

7.6 Invalidation and Expiration

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

- Section 7.6.1, "Select the Invalidation Method Best Suited for Your Content to Keep Performance in Check"
- Section 7.6.2, "Build Programmatic Invalidation Into Application Logic to Invalidate Dynamic Content"
- Section 7.6.3, "Combine Invalidation and Expiration Policies to Keep Cache Content Fresh"
- Section 7.6.4, "Use Invalidation Propagation in Clusters to Improve Data Consistency"

7.6.1 Select the Invalidation Method Best Suited for Your Content to Keep Performance in Check

Pick an invalidation type that works best for your content, keeping performance in mind:

- Use basic invalidation for one object based on an exact URL.

  A basic invalidation does not result in a cache traversal, making a basic invalidation highly performant.

- Advanced invalidation for multiple objects that share one or more of the following common elements:
  - Path prefix
  - Host name and port number
  - HTTP request method
  - URL expression
  - `POST` body expression
  - Search keys from the `Surrogate-Key` response-header field
  - Cookies
  - HTTP request headers
  - Embedded URL parameters

  When OracleAS Web Cache receives an advanced invalidation request, it traverses the contents of the cache to locate the objects to invalidate. Depending on the structure and number of objects cached, it can take time for OracleAS Web Cache to invalidate content.

  For the quickest advanced invalidation, specify a substring match instead of regular expression match.

- Inline invalidation to support either basic or advanced invalidation requests from the HTTP response, marked with the ESI `<esi:invalidate>` tag.

ORCL00528175

Inline invalidation reduces the connection overhead associated with sending out-of-band invalidations.

Implementation Details

To send a basic invalidation request:

- Use the step-by-step Invalidation wizard provided in Application Server Control Console. To get started:
  1. Navigate to the **Web Cache Home** page > **Administration** tab > **Operations** > **Invalidation**.
  2. In the first page of the Invalidation wizard, select **The single object that matches this URL**.
  3. Enter the complete URL patch and file name in the **Object to Invalidate** field.
- For a standalone OracleAS Web Cache installation, use the OracleAS Web Cache Manager.

  To get started, navigate to **Operations** > **Basic Content Invalidation.**
- Use the `BASICSELECTOR` element in a manual invalidation request

To send an advanced invalidation request:

- Use the step-by-step Invalidation wizard provided in Application Server Control Console. To get started:
  1. Navigate to the **Web Cache Home** page > **Administration** tab > **Operations** > **Invalidation**.
  2. In the first page of the Invalidation wizard, select **Specified Objects**.
- For a standalone OracleAS Web Cache installation, use the OracleAS Web Cache Manager.

  To get started, navigate to **Operations** > **Advanced Content Invalidation.**
- Use the `ADVANCEDSELECTOR` element in a manual invalidation request.

See Also: Chapter 13, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide* for further information about sending basic and advanced invalidation requests, as well as inline invalidations

7.6.2 Build Programmatic Invalidation Into Application Logic to Invalidate Dynamic Content

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`. 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.

ORCL00528176

Similarly, you can build invalidation into PL/SQL applications using the OracleAS Web Cache PL/SQL invalidation API `wxvutil.sql` and `wxvappl.sql`. This way, developers can embed the invocation of invalidation PL/SQL procedure calls into the PL/SQL Web page.

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.

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.

Implementation Details

See Also:

- Chapter 13, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide*
- *Oracle Application Server Containers for J2EE JSP Tag Libraries and Utilities Reference*. for further information about JESI

7.6.3 Combine Invalidation and Expiration Policies to Keep Cache Content Fresh

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

With invalidation, OracleAS Web Cache marks objects as invalid. When objects are marked as invalid and a client requests them, they are removed and then refreshed with new content from the origin servers. You can choose to remove and refresh invalid objects immediately, or base the removal and refresh on the current load of the origin servers. You can automate invalidation policies, as described in Section 7.6.2, "Build Programmatic Invalidation Into Application Logic to Invalidate Dynamic Content" on page 7-12.

7.6.4 Use Invalidation Propagation in Clusters to Improve Data Consistency

In a cache cluster, 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.

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.

ORCL00528177

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.

Implementation Details

> **See Also:** Chapter 13, "Sending Invalidation Requests," in the *Oracle Application Server Web Cache Administrator's Guide*

7.7 Optimizing Response Times

The following sections describe best practices in optimizing response times:

- Section 7.7.1, "Tuning Origin Server and OracleAS Web Cache Settings to Optimize Response Time"
- Section 7.7.2, "Use Compression to Improve Response Times and Reduce Network Bandwidth"
- Section 7.7.3, "Use Only Warning or Notification Logging Levels to Conserve Resources"

7.7.1 Tuning Origin Server and OracleAS Web Cache Settings to Optimize Response Time

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.

Implementation Details

Check the values of the network-related parameters in OracleAS Web Cache and the application Web server configuration file (`httpd.conf`).

> **See Also:** Section 7.6, "Tune Network-Related Parameters," in the *Oracle Application Server Performance Guide* contains information on how to set turning parameters in OracleAS Web Cache and the application server

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 objects a higher priority than the cacheable objects.
- 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 7.4.2, "Allocate Sufficient Network Bandwidth to Accommodate the Throughput Load" on page 7-6.
- CPU usage. See Section 7.2.1, "Use Two CPUs and Consider Deploying on Dedicated Hardware to Avoid Operating System Limitations" on page 7-2.

ORCL00528178

- 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 Section 7.6, "Tune Network-Related Parameters," in the *Oracle Application Server Performance Guide*.

7.7.2 Use Compression to Improve Response Times and Reduce Network Bandwidth

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 objects.

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 or no option that an administrator selects when specifying a caching rule. Because OracleAS Web Cache supports regular expression for caching rules, you can apply compression 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, 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.

Implementation Details

See Also: Chapter 12, "Creating Caching Rules," in the *Oracle Application Server Web Cache Administrator's Guide* for instructions about how to enable compression, as well as a complete listing of objects not compressed by OracleAS Web Cache

7.7.3 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. 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 fail. 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.

ORCL00528179

Implementation Details

See Also: Chapter 15, "Using Diagnostics Tools," in the *Oracle Application Server Web Cache Administrator's Guide* for more information about logging

ORCL00528180

Exhibit 26

UNITED STATES DEPARTMENT OF COMMERCE
Patent and Trademark Office
Address: COMMISSIONER OF PATENTS AND TRADEMARKS
Washington, D.C. 20231

cu

MF

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.
08/794,269	02/03/97	ADUNUTHULA S	3010-060

LMC1/0830
HICKMAN PALERMO TRUONG & BEZKER
1600 WILLOW STREET
SAN JOSE CA 95125-5106

EXAMINER
LAO,S

ART UNIT	PAPER NUMBER
2755	19

DATE MAILED: 08/30/00

Please find below and/or attached an Office communication concerning this application or proceeding.

Commissioner of Patents and Trademarks

PTO-90C (Rev. 2/95) U.S. G.P.O. 1999 460-693 1- File Copy

TD

Office Action Summary

Application No.	Applicant(s)
08/794,269	Adunuthula, et al
Examiner	**Group Art Unit**
S. Lao	2755

[X] Responsive to communication(s) filed on *Jun 8, 2000*

[ ] This action is **FINAL.**

[ ] Since this application is in condition for allowance except for formal matters, **prosecution as to the merits is closed** in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11; 453 O.G. 213.

A shortened statutory period for response to this action is set to expire *3* month(s), or thirty days, whichever is longer, from the mailing date of this communication. Failure to respond within the period for response will cause the application to become abandoned. (35 U.S.C. § 133). Extensions of time may be obtained under the provisions of 37 CFR 1.136(a).

Disposition of Claim

[X] Claim(s) *1-12 and 14-36* is/are pending in the applicat

Of the above, claim(s) _____ is/are withdrawn from consideration

[ ] Claim(s) _____ is/are allowed.

[X] Claim(s) *1-12 and 14-36* is/are rejected.

[ ] Claim(s) _____ is/are objected to.

[ ] Claims _____ are subject to restriction or election requirement.

Application Papers

[ ] See the attached Notice of Draftsperson's Patent Drawing Review, PTO-948.

[ ] The drawing(s) filed on _____ is/are objected to by the Examiner.

[ ] The proposed drawing correction, filed on _____ is [ ] approved [ ] disapproved.

[ ] The specification is objected to by the Examiner.

[ ] The oath or declaration is objected to by the Examiner.

Priority under 35 U.S.C. § 119

[ ] Acknowledgement is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d).

[ ] All [ ] Some* [X] None of the CERTIFIED copies of the priority documents have been

[ ] received.

[ ] received in Application No. (Series Code/Serial Number) _____.

[ ] received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

*Certified copies not received: _____

[ ] Acknowledgement is made of a claim for domestic priority under 35 U.S.C. § 119(e).

Attachment(s)

[X] Notice of References Cited, PTO-892

[ ] Information Disclosure Statement(s), PTO-1449, Paper No(s). _____

[ ] Interview Summary, PTO-413

[ ] Notice of Draftsperson's Patent Drawing Review, PTO-948

[ ] Notice of Informal Patent Application, PTO-152

--- SEE OFFICE ACTION ON THE FOLLOWING PAGES ---

ORCL01623264

DETAILED ACTION

1. Claims 1-12, 14-36 are pending. This action is in response to applicant's response filed 6/8/2000.

2. The text of those sections of Title 35, U.S. Code not included in this action can be found in a prior Office action.

3. Claims 1, 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al in view of Heizer and Kahn et al.

As to claim 1, Putz teaches (abstract, col. 9, lines 21-30; col. 10, lines 20-37; col. 12, lines 33-55) a method for responding to a request (remote procedure calls) issued to a server (main server program 55) from a client (client 52-54) over a network system, the method comprising the steps of:

the server obtaining the request over a network (receive remote procedure calls);

the server identifying (check program dictionary) a program type corresponding to the request (named database operation program) among a plurality of program types (operation programs NewDocDecs, Render, DescriptionSearch),

if no currently executing instance of the program type (named operation program) exists, selectively initiating an instance of the program type (fork a child process of the named operation program),

the server dispatching the request to the instance (fork, then pass the remaining operating program arguments),

executing the instance to process the request (invoke operating program for performing),

responding to the request (return results of the operation).

Putz does not teach (1) the step of initiating is based on a comparison between how many instances are executing of the program type is executing and a prescribed number

of the program type, (2) determining whether an instance of the program type currently exists that is not currently processing a previously received request.

As to (1), Heizer teaches a method for responding to a request (request) issued to a server program type (server 122) from a client (client 103), including initiating an instance of the program type (server process 202, 203) based on a comparison between how many instances are executing of the program type is executing (fig. 3) and a prescribed number of instances of the program (low/high water mark); see col. 2, line 20 - col. 4, line 55; fig.s 1-5.

Putz creates an instance of a named program type for each client request of that named program type, which eventually will cause server overloading. This the aspect of the request dispatching that Heizer addresses to improve. (Col. 1, lines 21-41). Therefore, one of the ordinary skill in the art would have been motivated to apply the teaching of Heizer to Putz's dispatching of each named program, to prevent server overloading. The combination would have provided maximum number of instances for each named program type.

As to (2), Kahn teaches management of server resource (internal reader), wherein if there is a server instance currently exists (previously created internal reader) that is not currently processing a previously received request (which has finished processing and waiting for additional work, idle), an incoming request/job is assigned to such server instance. Otherwise, new server instances are initiated/created to handle the request/job. See abstract, col. 13, lines 40-63, col. 15, lines 35-61, col. 18, lines 11-35. Kahn determines whether there is an available/idle/not_currently_processing server instance by checking the IDLE flag for an internal reader. It would have been obvious to include the teaching of Kahn into Putz as modified so as to provide handling of changing work flow without excessive system overhead (Kahn, col. 18, lines 31-35).

As to claim 25, it is program product claim of claim 1.

4. Claims 2-3 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al and further in view of Donica et al.

As to claim 2, Donica teaches responding to a client request, including obtaining the request by a server process (scheduler) executing in a first address space (scheduler address space 30), initiating an instance of a service program (initiator) in a second address space that is separate from the first address space (initiator address space 40 separate from 30), see col. 5, lines 31-34. Since Putz, Heizer, Kahn et al and Donica address load balancing and reducing operating system overhead, it would have been obvious to combine the teachings.

As to claim 3, Donica teaches after executing an instance, maintaining it in memory for executing a subsequent request (idle initiators already assigned to a class), see col. 7, lines 19-24. Note discussion of claim 2 for motivation to combine.

5. Claims 4, 26 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claims 1, 25 and further in view of Tobe et al.

As to claim 4, Tobe teaches network resource management, including detecting (detect by the system) a fault in an instance during execution thereof (node goes down); and aborting the instance in response to detecting the fault (back up node takes over), see col. 9, lines 42-52. Since Putz, Heizer, Kahn et al and Tobe address resource management and overhead, it would have been obvious to combine the teachings.

As to claim 26, it is program product claim of claim 4.

6. Claims 5-9, 27 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claims 1, 25 and further in view of Donica et al and Atkins et al.

As to claim 5, Donica teaches specifying at least one of a maximum (maximum number of initiators) and a minimum (minimum number of initiators) number of instances (col. 6, lines 3-5, fig. 2).

As to that each instance executes within an address space that is separate from the address spaces used by other instances of the program, it is taught by Atkins in that each node holds an instance (copy) of a service program (shared object), see pg. 3, left col.,

lines 7-20. Since Putz, Heizer, Donica and Atkins address server resource management, it would have been obvious to combine the teachings.

As to claims 6-9, Heizer teaches initiating a specified minimum number of instances (one server process 202) prior to obtaining the request over the network (on start up, col. 3, lines 16-17), after obtaining the request over the network, determining if one of the minimum number of instances is available for processing said request (first request, col. 4, lines 3-13), after obtaining the request over the network, initiating a new instance of the program if none of the minimum number of instances is available for processing said request and the number of instances is less than the maximum number of instances (third request, col. 4, lines 17-29), executing said instance includes executing an available instance from the minimum number of instances (first request, col. 4, lines 3-13), returning the request to the network without processing said request based on the prescribed number of said instances exceeding a maximum value (21st request, col. 4, lines 52-56).

As to claim 27, it is program product claim of claim 5.

7. Claims 10-11 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claim 1 and further in view of Donica et al and Travis et al.

As to claims 10-11, Donica teaches specifying a maximum number of instances, as discussed on claim 5. Travis teaches registering a plurality of programs (applications) with the server (register, col. 17, line 58 - col 18, line 45; server registration, col. 18), specifying a virtual path for a program (names of classes/messages), request (invocation, message) identifies a virtual path (message names), identifying the program (method to perform the operation) based on the virtual path identified in the request (select from the database, Invoker Operation, col.s 25-29), Since Putz, Heizer, Kahn et al, Donica and Travis address request allocation, it would have been obvious to combine the teachings.

Travis teaches specifying other data such as performance information (parameters supported, col. 18, lines 20-29), it would have been obvious to specify a maximum number of instances which is commonly regarded as performance information.

8. Claims 12, 21-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer and Kahn et al as applied to claim 1 and further in view of Djakovic.

As to claim 12, Heizer teaches receiving the request from a transport protocol process (protocol/network software 113), but fails to teach receiving is according to a first protocol and converting the request to a second protocol independent from the first protocol.

However, Djakovic teaches receiving a request (job submitted by client) according to a first protocol (source protocol) and converting the request to a second protocol independent from the first protocol (form a new job in target protocol), see col. 1, lines 40-51; fig. 3. Since Putz, Heizer, Kahn et al and Djakovic address server processing clients request/job, it would have been obvious to combine the teachings.

As to claim 21, note discussion of claim 1 for obtaining, step A)-C), and note discussion of claim 12 for a transport adaptor / means configured to perform protocol conversions. Heizer further teaches forwarding the request to a dispatcher (control process 204 accepts new client) executed by the server; and processing the request by causing the dispatcher to perform the step D) if no instance is available and the existing number of instances exceeds the maximum prescribed number, then sending said reply over the network indicating the request was not processed (when the maximum number of server processes is set at 20, request 21st is rejected, step 317), see col. 3, lines 10-23; col. 4, lines 3-56. Note discussions of claims 1, 12 for motivations to combine.

As to claim 22, note discussion of claim 6.

As to claim 23, Heizer teaches sending a replying indicating the request is not processed (reject, step 317), and sending such a reply when no program configured to handle the request would have been an obvious condition to apply.

As to claim 24, Heizer teaches server process (control process 204), and plug-in routine added to the server process (library 121) .

9. Claims 28-30, 34, 36 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al as applied to claim 1 and further in view of admitted prior art (pages 1-3).

As to claim 28, note discussion of claim 1, in particular, note claim 1 for receiving request and sending response, for dispatcher selectively dispatching / dispatching and sending the response / responding, for not currently processing and each program having a prescribed number of instances / maximum / minimum. The instances (server process) of a program type of Heizer reside at respective address spaces (202, 203) since they execute on their respective clients.

Further, Heizer teaches network listener (protocol/network software 113), each program type configured to perform an operation that generates an output in response to receiving a request (a server process is to receive a client request, process the request and send a response).

As to that the dispatcher is in a plug-in format, it is taught by admitted prior art (plug-in extensions, page 2, last para.). It would have been obvious to include the plug-in format of admitted prior art into the system/method of Heizer as modified to provide dynamic response to request (page 2, last para.).

As to claims 29-30, admitted prior art teaches receiving the request and sending the response based on Hypertext Transfer Protocol (HTTP) (HTTP, page 1, lines 9-10), to output a static Hypertext Markup Language (HTML) page (page 1, last para.). A daemon is a well known means for performing services.

As to claim 34, Heizer teaches initiating a prescribed minimum number of said instances of the corresponding program (one instance, see rejection of claim 13). Each execution engine is taught by well known distributed network managers.

As to claim 36, Putz teaches registering (the program directory) program types (named database operations) with a dispatcher (main server program 55) for dispatching client requests. (col. 10, lines 31-37). Typically registering with a program directory includes registering the locations/addresses.

10. Claim 35 is rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of admitted prior art as applied to claim 28 and further in view of Donica.

As to claim 35, Donica teaches configuration library (PARMLIB) identifying for each of the programs the corresponding object type (class) and the prescribed number of instances (maximum and minimum numbers of initiators), see col. 6, lines 1-5; col. 8, lines 27-31. As to plug-in format, note discussion of claim 28. Note rejection of claim 2 for motivation to combine.

11. Claim 31 is rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of admitted prior art as applied to claim 28 and further in view of Djakovic.

As to claim 31, note the discussion of claim 12.

12. Claim 32-33 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of admitted prior art as applied to claim 28 and further in view of Page et al.

As to claims 32-33, Page teaches a plurality of execution engines to control the execution of program instances and to receive the request (adapters to DCE, DDE, LU6.2) (distributed brokers), API specifying predetermined operations, including at least one of initialization (REGISTER), execution of the request (SEND), and shutdown (EOC, ABENDS), see fig.s 8, 17B, 21, 23. Since Putz, Heizer, Donica, Atkins, APA, Travis and Page address client server management, it would have been obvious to combine the teachings.

13. Claims 14, 17 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al as applied to claim 1 and further in view of Donica et al.

As to claim 17, note discussions of claims 1, 10-11 (with respect to Donica), and Heizer further teaches forwarding the request to a dispatcher (control process 204 accepts

new client) executed by the server; and processing the request by causing the dispatcher to perform the steps of A) determining whether an available instance of a program configured to handle the request is available among an existing number of instances of the program (decides which of 202, 203 should handle the client), B) if an available instance is available, then dispatching the request for execution by the available instance (first request is given to 202 since it is activated, step 307), C) if no instance is available, then initiating a new instance of the program for execution of the request if the existing number of instances does not exceed a maximum prescribed number (for the third request, a new server process 203 is started, step 313), and D) if no instance is available and the existing number of instances exceeds the maximum prescribed number, then sending said reply over the network indicating the request was not processed (when the maximum number of server processes is set at 20, request 21st is rejected, step 317), see col. 3, lines 10-23; col. 4, lines 3-56.

Further, Putz teaches registering (the program directory) program types (named database operations) with a dispatcher (main server program 55) for dispatching client requests. (col. 10, lines 31-37). Typically registering with a program directory includes registering the locations/addresses. Registering with a dispatcher (control process) information of a maximum number of instances (high water mark) of a service is taught by Heizer (see discussion of claim 1).

As to claim 14, Heizer teaches receiving by the dispatcher (control process/server program) a reply (response) from the instance executing and sending information contained to a client (send response). See fig. 1.

14. Claims 15-16 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of Donica et al as applied to claim 17 and further in view of Tobe et al.

As to claim 15, note discussion of claim 4 for detecting and terminating/aborting and dispatcher (management node, col. 4, lines 49-51).

As to claim 16, sending a reply is covered by claim 17, step D).

15. Claims 18-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Putz et al, Heizer, Kahn et al in view of Donica et al as applied to claim 17 and further in view of Atkins et al.

As to claims 18-19, note discussion of claim 5.

16. Claim 20 is rejected under 35 U.S.C. 103(a) as being unpatentable over Putz, Heizer, Kahn in view of McChesney et al.

As to claim 20, note discussions of claims 1, and Heizer further teaches forwarding the request to a dispatcher (control process 204 accepts new client) executed by the server; and processing the request by causing the dispatcher to perform the steps of A) determining whether an available instance of a program configured to handle the request is available among an existing number of instances of the program (decides which of 202, 203 should handle the client), B) if an available instance is available, then dispatching the request for execution by the available instance (first request is given to 202 since it is activated, step 307), C) if no instance is available, then initiating a new instance of the program for execution of the request if the existing number of instances does not exceed a maximum prescribed number (for the third request, a new server process 203 is started, step 313), and D) if no instance is available and the existing number of instances exceeds the maximum prescribed number, then sending said reply over the network indicating the request was not processed (when the maximum number of server processes is set at 20, request 21st is rejected, step 317), see col. 3, lines 10-23; col. 4, lines 3-56.

As to E), McChesney teaches a process/object life cycle management system, including delaying and automatically deallocating an instance after a predetermined time interval (API Shutdown 223, automatically shutdown an idle process after a time interval). See fig. 2, col. 30, line 52 - col. 31, line 3. Since McChesney addresses object/resource management as does Putz as modified, it would have been obvious to combine the teachings.

17. Applicant's arguments filed 6/8/2000 have been considered but are moot in view of the new ground(s) of rejection.

In the response, applicant's representative argued in substance that the motivations to combine the cited references must be expressly found in the cited references and not based on hindsight/applicant's disclosure.

The examiner's response is as follows.

MPEP § 2143.01 requires that the suggestion or motivation to combine references must be found in the references themselves, OR, in the knowledge generally available to one of ordinary skill in the art. The proper test for the relevance of a cited combination of references is: "whether the teachings of the prior art, taken as a whole, would have made obvious the claimed invention," In re Gorman, 933 F.2d at 986, 18 USPQ2d at 1888. Subject matter is unpatentable under section 103 if " 'would have been obvious ... to a person having ordinary skill in the art.' While there must be some teaching, reason, suggestion, or motivation to combine existing elements to produce the claimed device, it is not necessary that the cited references or prior art specifically suggest making the combination: In re Nilssen, 851 F.2d 1401, 1403, 7 USPQ2d 1500, 1502 (Fed. Cir. 1988)." Such suggestion or motivation to combine prior art teachings can derive solely from the existence of a teaching, which one of ordinary skill in the art would be presumed to know, and the use of that teaching to solve the same [or] similar problem which it addresses. In re Wood, 599 F.2d 1032, 1037, 202 USPQ 171, 174 (CCPA 1979). "In stun, it is off the mark for litigants to argue, as many do, that an invention cannot be held to have been obvious unless a suggestion to combine prior art teachings is found in a specific reference." In re Oetiker, 24 USPQ2d 1443 (CAFC 1992).

In response to Applicant's argument that the Examiner's conclusion of obviousness is based upon improper hindsight reasoning, it must be recognized that any judgement on obviousness is in a sense necessarily a reconstruction based upon hindsight reasoning. But so long as it takes into account only knowledge which was within the level of ordinary skill at the time the claimed invention was made, and does not include knowledge gleaned

only from the applicant's disclosure, such a reconstruction is proper. In re McLaughlin, 443 F.2d 1392; 170 USPQ 209 (CCPA 1971).

18. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Sue Lao whose telephone number is (703) 305-9657. The fax number for this Group is (703) 305-9731.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the Group receptionist whose telephone number is (703) 305-3900.

Sue Lao
August 23, 2000

MAJID BANANKHAH
PRIMARY EXAMINER

Notice of References Cited

Application No.	Applicant(s)		
08/794,269	Adunuthula, et al		
Examiner		Group Art Unit	Page 1 of 1
S. Lao		2755	

U.S. PATENT DOCUMENTS

	DOCUMENT NO.	DATE	NAME	CLASS	SUBCLASS
A	5,894,554	4/1999	Lowery et al	395	200.33
B	6,070,191	5/2000	Narendran et al	709	226
C	6,098,093	8/2000	Bayeh et al	709	203
D					
E					
F					
G					
H					
I					
J					
K					
L					
M					

FOREIGN PATENT DOCUMENTS

	DOCUMENT NO.	DATE	COUNTRY	NAME	CLASS	SUBCLASS
N						
O						
P						
Q						
R						
S						
T						

NON-PATENT DOCUMENTS

	DOCUMENT (Including Author, Title, Source, and Pertinent Pages)	DATE
U		
V		
W		
X		

ORCL01623276

Exhibit 27

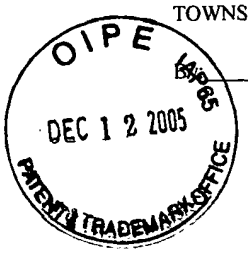
I hereby certify that this correspondence is being deposited with the United States Postal Service as first class mail in an envelope addressed to:

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

On December 9, 2005

TOWNSEND and TOWNSEND and CREW LLP

By



PATENT
Attorney Docket No.: 021756-015800US
Client Reference No.: OID-2005-241-01

2155
TPW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Examiner: Philip B. TRAN
Marc S. CASALAINA	Art Unit: 2155
Application No.: 09/615,383	INFORMATION DISCLOSURE STATEMENT UNDER 37 CFR §1.97 and §1.98
Filed: July 13, 2000	
For: Delta Caching	

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

The references cited on attached form PTO/SB/08A&B are being called to the attention of the Examiner. Copies of the references [in compliance with the requirements of 1287 OG 163] are enclosed.

It is respectfully requested that the cited references be expressly considered during the prosecution of this application, and the references be made of record therein and appear among the "references cited" on any patent to issue therefrom.

As provided for by 37 CFR 1.97(g) and (h), no inference should be made that the information and references cited are prior art merely because they are in this statement and no

12/13/2005 MBERHE 00000038 201430 09615383
01 FC:1806 180.00 DA

representation is being made that a search has been conducted or that this statement encompasses all the possible relevant information.

This IDS is being filed before the mailing date of the final Office Action or Notice of Allowance.

Please charge the IDS fee of $180 to Deposit Account No. 20-1430. Please deduct any additional fees from, or credit any overpayment to, the above-noted Deposit Account.

Respectfully submitted,

Dated: 12/9/05

Brian N. Young
Reg. No. 48,602

TOWNSEND and TOWNSEND and CREW LLP
Two Embarcadero Center, Eighth Floor
San Francisco, California 94111-3834
Tel: 415-576-0200
Fax: 415-576-0300
BNY/

60643635 v1

DEC 12 2005 IAP65 PATENT & TRADEMARK OFFICE

PTO/SB/08A&B (08-03)

Substitute for form 1449A&B/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(use as many sheets as necessary)

Sheet 1 of 3

Complete if Known	
Application Number	09/615,383
Filing Date	July 13, 2000
First Named Inventor	Marc S. CASALAINA
Art Unit	2155
Examiner Name	Philip B. TRAN
Attorney Docket Number	021756-015800US

U.S. PATENT DOCUMENTS+

Examiner Initials*	Cite No.[1]	Document Number Number Kind Code[2] (*if known*)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear
	AA	US-5,873,069	02-16-1999	Reuhl et al.	
	AB	US-5,870,544	02-09-1999	Curtis	
	AC	US-5,894,554	04-13-1999	Lowery et al.	
	AD	US-5,945,989	08-31-1999	Freishtat et al.	
	AE	US-6,052,730	04-18-2000	Felciano et al.	
	AF	US-6,092,074	07-18-2000	Rodkin et al.	
	AG	US-6,105,004	08-15-2004	Halperin et al.	
	AH	US-6,122,661	09-19-2000	Stedman et al.	
	AI	US-6,226,642	05-01-2001	Britton et al.	
	AJ	US-6,226,792	05-01-2001	Goiffon et al.	
	AK	US-6,263,352	07-17-2001	Cohen	
	AL	US-6,266,684	07-24-2001	Kraus et al.	
	AM	US-6,308,275	10-23-2001	Vaswani	
	AN	US-6,415,319	07-02-2002	Ambroziak	
	AO	US-6,434,578	08-13-2002	McCauley et al.	
	AP	US-6,438,578	08-20-2002	Schmid et al.	
	AQ	US-6,457,030	09-24-2002	Adams et al.	
	AR	US-6,493,679	12-10-2002	Rappaport et al.	
	AS	US-6,594,691	07-15-2003	McCollum et al.	
	AT	US-6,601,057	07-29-2003	Underwood et al.	
	AU	US-6,668,354	12-23-2003	Chen et al.	
	AV	US-6,760,750	07-06-2004	Boneh et al.	

FOREIGN PATENT DOCUMENTS

Examiner Initials*	Cite No.[1]	Foreign Patent Document Country Code[3] Number[4] Kind Code[5] (*if known*)	Publication Date MM-DD-YYYY	Name of Patentee or Applicant of Cited Document	Pages, Columns, Lines, Where Relevant Passages or Relevant Figures Appear	T[6]
						☐

NON PATENT LITERATURE DOCUMENTS

Examiner Initials *	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T[2]
	AW	JENKS, KEN, Re: "Re: JavaScript Calling CGI Script," in comp.lang.javascript, February 16, 2000, downloaded from < http://groups.google.com/group/comp.lang.javascript/msg/343ea4daa298e8df?hl=en&>	
	AX	QUANTZ, LARRY, "need help with 'merging' two html files together," in comp.lang.javascript, February 6, 2000, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/1f890780cb6c22e6?hl=en&>	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

OIPE IAP65 DEC 12 2005 PATENT & TRADEMARK OFFICE

PTO/SB/08A&B (08-03)

Substitute for form 1449A&B/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(use as many sheets as necessary)

Sheet 2 of 3

Complete if Known	
Application Number	09/615,383
Filing Date	July 13, 2000
First Named Inventor	Marc S. CASALAINA
Art Unit	2155
Examiner Name	Philip B. TRAN
Attorney Docket Number	021756-015800US

NON PATENT LITERATURE DOCUMENTS

Examiner Initials *	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T[2]
	AY	WILKINSON, BILL, "Re: Response Redirect," in Microsoft.public.inetserver.asp.general, February 3, 2000, downloaded from <http://groups.google.com/group/microsoft.public.inetserver.asp.general/msg/8dd026d00167fd14?hl=en&>	
	AZ	JENKS, KEN, "Re: Submitting all forms within a HTML-page?" in comp.lang.javascript, November 22, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/d4654b5ac56a42b2?hl=en&>	
	BA	BEILBY, JAMES, "Re: Help me I'm dumb" in comp.lang.javacript, September 18, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/8e7c1ab6c5868e48?hl=en&>	
	BB	PAYNE, PATRICK, "Re: Web Development - the future or a sideline?" in comp.lang.javascript, August 23, 1999, downloaded from <http://groups.google.com/group/comp.databases.pick/msg/e55f8bb165bbec36?hl=en&>	
	BC	ZIGRAY, DAVID, "Web Development – the future or a sideline? – NOW!" in comp.databases.pick, August 25, 1999, downloaded from <http://groups.google.com/group/comp.databases.pick/msg/772b10140e56fe47?hl=en&>	
	BD	STORMS, IAN A., "Re: Connecting to Database!!!" in comp.lang.javascript, August 17, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/5536270d890750fc?hl=en&>	
	BE	GREENWOOD, TONY, "Re: Help! Passing vars between Perl and JavaScript," in comp.lang.javascript, May 18, 1999, downloaded from "http://groups.google.com/group/comp.lang.javascript/msg/d6d6e18ddd89b282?hl=en&"	
	BF	GREENWOOD, TONY, "Re: receiving parameters," in comp.lang.javascript, March 22, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/88b9dd5f7f502e0d?hl=en&>	
	BG	GREESON, ROY et al., "Is this possible with Client-Side Java?" in comp.lang.javascript, March 20, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/browse_thread/thread/3788f12a16af547f/2446b8a8bd23e1aa#2446b8a8bd23e1aa>	
	BH	STORMS, IAN A., "Re: the efficiency of JavaScript Variables," in comp.lang.javascript, January 25, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/5912db735c7685e8?hl=en&>	
	BI	SIGIROS, ANGELO, "Re: passing data: frame1>server>frame2 how???" in netscape.devs-javascript, January 8, 1999, downloaded from <http://groups.google.com/group/netscape.devs-javascript/msg/921acdc6f9d30ce4?hl=en&>	
	BJ	HONNEN, MARTIN, "Re: JavaScript polling a servlet/cgi," in borland.public.intrabuilder.javascript, October 9, 1998, downloaded from <http://groups.google.com/group/borland.public.intrabuilder.javascript/msg/4b70e4bc473fc93a?hl=en&>	
	BK	ZANDSTRA, MATT, "Re: passing variables," in comp.lang.javascript, August 12, 1998, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/c8f282bf26253864?hl=en&>	
	BL	BRODERICK, ANDREW, "Re: Sub Drop Down List," in microsoft.public.vinterdev, July 1, 1998, downloaded from <http://groups.google.com/group/microsoft.public.vinterdev/msg/74b1b9fafd6e9f89?hl=en&>	
	BM	FREEDMAN, RICHARD, "Re: Passing complex structure (binary tree) from java to javascript," in comp.lang.javascript, April 16, 1998, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/84988957fc822ec5?hl=en&>	
	BN	VALLEY, THOMAS, "Re: Opening a URL in the background," in netscape.devs-javascript, March 19, 1998, downloaded from <http://groups.google.com/group/netscape.devs-javascript/msg/72126c9d8cc41723?hl=en&>	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

DEC 12 2005
PATENT & TRADEMARK OFFICE

PTO/SB/08A&B (08-03)

Substitute for form 1449A&B/PTO

INFORMATION DISCLOSURE STATEMENT BY APPLICANT

(use as many sheets as necessary)

Sheet 3 of 3

Complete if Known	
Application Number	09/615,383
Filing Date	July 13, 2000
First Named Inventor	Marc S. CASALAINA
Art Unit	2155
Examiner Name	Philip B. TRAN
Attorney Docket Number	021756-015800US

NON PATENT LITERATURE DOCUMENTS

Examiner Initials *	Cite No.[1]	Include name of the author (in CAPITAL LETTERS), title of the article (when appropriate), title of the item (book, magazine, journal, serial, symposium, catalog, etc.), date, page(s), volume-issue number(s), publisher, city and/or country where published.	T[2]
	BO	SEGURA, ALVARO, 'Re: Client JScript talking to Server," in comp.lang.javascript, March 18, 1999, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/0afe1f53f594aa1b?hl=en&>	
	BP	JENKS, KEN, "Re: Can you call a Java Servlet through javascript?" in comp.lang.javascript, January 6, 2000, downloaded from <http://groups.google.com/group/comp.lang.javascript/msg/249ef5feb780b97e?hl=en&>	
	BQ	NEWTON, STUART, "Re: Problem on variables – Need your help!" in microsoft.public.inetserver.iis.activeserverpages, January 12, 2000, downloaded from <http://groups.google.com/group/microsoft.public.inetserver.iis.activeserverpages/msg/45daf823b1585941?hl=en&>	
	BR	CAMDEN, RAY, "Creating a 'Live' ColdFusion Site," ColdFusion Developer's Journal, January 12, 1999, downloaded from <http://cfdj.sys-con.com/read/41481.htm>	
	BS	JALIX.ORG, "The Dynamic Duo – Cross-Browser Dynamic HTML," November 11, 1999, downloaded from <http://www.jalix.org/ressources/internet/dhtml/_dynduo/dynduo-19991024.pdf>	
	BT	Schwartz, "Chapter 20 – VBScript Forms, Controls, and Managing Transactions," Special Edition Using Visual Basic Script, QUE Corporation, 1996, downloaded from < http://www.urc.bl.ac.yu/manuals/vbscript/ch20fi.htm>	
	BU	Baron, J., & M. Siepmann, "Techniques for Creating and Using Web Questionnaires in Research and Teaching," in Psychological Experiments on the Internet (M. H. Birnbaum (Ed.) (2000), downloaded from <http://www.psych.upenn.edu/~baron/examples/baron4.htm>	
	BV	Remote Scripting, "http://msdn.microsoft.com/scripting/remotescripting..." Microsoft Corporation (January 4, 2000)	

Examiner Signature		Date Considered	

*EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609. Draw line through citation if not in conformance and not considered. Include copy of this form with next communication to applicant.

[1] Applicant's unique citation designation number (optional). [2] Applicant is to place a check mark here if English language Translation is attached.

Exhibit 28

Oracle® Web Cache

Administration and Deployment Guide

Release 1.0.2

November 21, 2000

Part No. A86722-01

ORACLE

Oracle Web Cache Administration and Deployment Guide, Release 1.0.2

Part No. A86722-01

Copyright © 2000, Oracle Corporation. All rights reserved.

Primary Author: Deborah Steiner

Contributors: Henry Abrecht, Steve Andrew, Jesse Anton, Pierre Baudin, Omar Bellal, Pramodini Gattu, Larry Jacobs, Xiang Liu, Shehzaad Nakhoda, Jean Zeng, Tie Zhong

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; 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 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. Oracle Corporation does not warrant that this document is 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, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions 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 Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle is a registered trademark and Oracle9*i* and Oracle8*i* are trademarks of Oracle Corporation. Other names may be trademarks of their respective owners.

1

Introduction to Oracle Web Cache

This chapter describes the performance barriers faced by Web sites and introduces the technology which can provide a complete caching solution.

This chapter contains these topics:

- What is the Big Picture for Caching?
- Oracle's Solution to Web Site Performance Issues
- How Web Caching Works
- Benefits of Web Caching
- Features of Oracle Web Cache

What is the Big Picture for Caching?

The e-business model creates new performance requirements for Web sites. To carry out electronic business successfully, Web sites must protect against poor response time and system outages caused by peak loads. Slow performance translates into lost revenue.

Many high-volume Web sites try to counter this problem by adding more application Web servers to their existing architecture. As more users access these Web sites, more and more application Web servers will have to be added. In short, the manageability costs associated with adding application Web servers often outweigh the benefits.

Static caches and content distribution services can provide some relief. However, these solutions are unable to serve content that is dynamically generated.

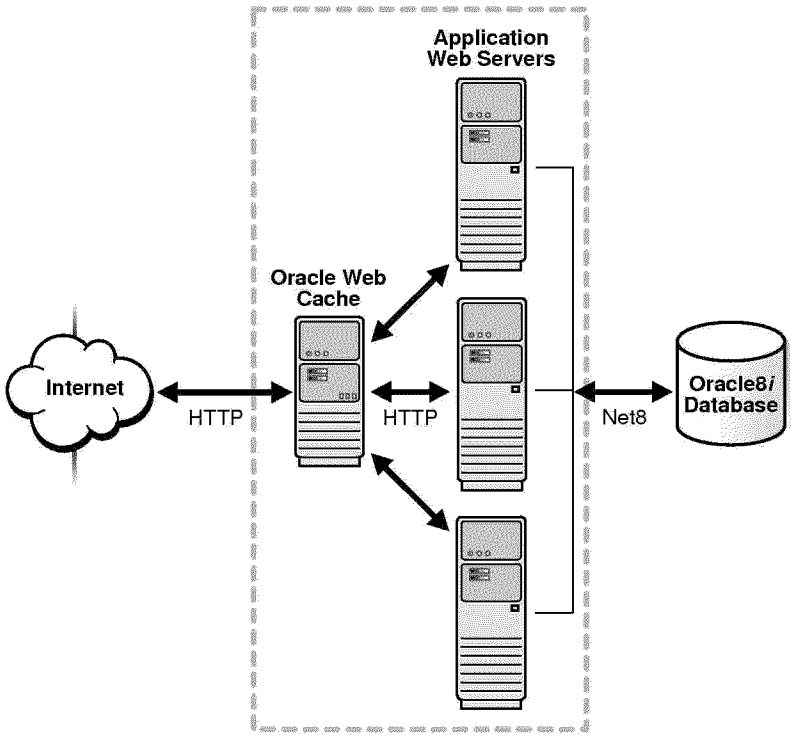
Oracle's Solution to Web Site Performance Issues

Faced with these performance challenges, e-businesses need to invest in more cost-effective technologies and services to improve the performance of their sites. Oracle offers Oracle Web Cache to help e-businesses manage Web site performance issues. Oracle Web Cache is a content-aware server accelerator, or **reverse proxy server**, that improves the performance, scalability, and availability of Web sites that run on Oracle9*i* Application Server and Oracle8*i*.

By storing frequently accessed URLs in virtual memory, Oracle Web Cache eliminates the need to repeatedly process requests for those URLs on the application Web server. Unlike legacy proxy servers that only handle static documents, Oracle Web Cache caches both static and dynamically generated content from one or more application Web servers. Oracle Web Cache retrieves content faster than legacy proxies and greatly reduces the load on application Web servers.

Figure 1–1 shows the basic architecture. Oracle Web Cache sits in front of application Web servers, caching their content, and providing that content to Web browsers that request it. When Web browsers access the Web site, they send HTTP requests to Oracle Web Cache. Oracle Web Cache, in turn, acts as a virtual server to the application Web servers. If the requested content has changed, Oracle Web Cache retrieves the new content from the application Web servers. The application Web servers may retrieve their content from an Oracle database.

Figure 1–1 Oracle Web Cache Architecture



How Web Caching Works

To Web browsers, Oracle Web Cache acts as the virtual server for application Web servers. You configure Oracle Web Cache with the same IP address that is registered for a site's domain name and the application Web servers' host names. This

configuration enables Web browsers to communicate with Oracle Web Cache rather than application Web servers when accessing a Web site.

Figure 1–2 on page 1-5 shows how Web caching works. Oracle Web Cache has an IP address of `144.25.190.240` and the application Web server has an IP address of `144.25.190.245`. The steps for browser interaction with Oracle Web Cache follow:

1. A browser sends a request to a Web site named `www.company.com`.

   This request in turn generates a request to Domain Name System (DNS) for the IP address of for the Web site.

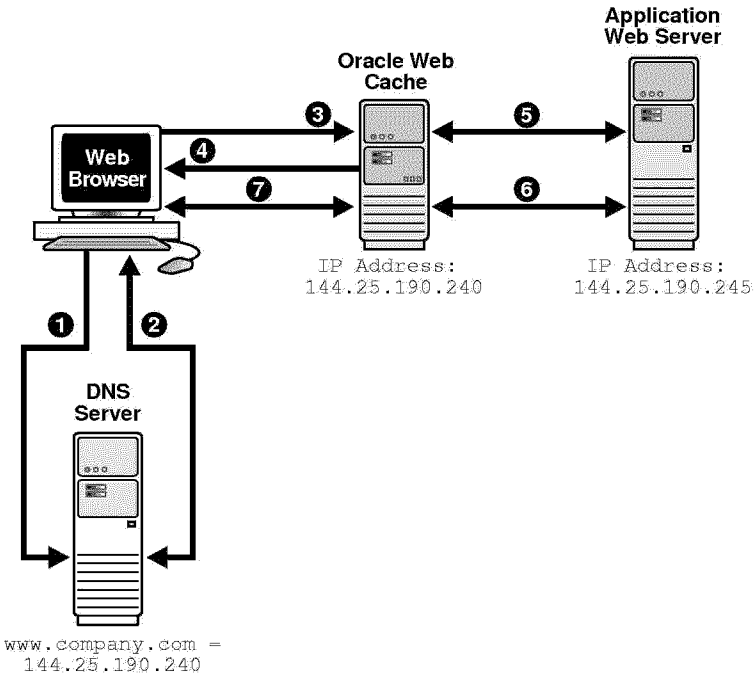
2. DNS returns the IP address of Oracle Web Cache, that is, `144.25.190.240`.
3. The browser sends the request for the Web page to Oracle Web Cache, `144.25.190.240`.
4. If the requested content is in its cache, Oracle Web Cache sends the content directly to the browser. This is called a **cache hit**.

   Note: Dynamic content is generated by the application Web server and then returned to Oracle Web Cache before being passed to the browser.

5. If Oracle Web Cache does not have the requested content or the content is stale or invalid, it hands the request off to the application Web server. This is called a **cache miss**.
6. The application Web server sends the content through Oracle Web Cache.
7. Oracle Web Cache sends the content to the client and makes a copy of the page in cache.

   Note: A page stored in the cache is removed when it becomes invalid or outdated, as described in "Cache Freshness and Performance Assurance" on page 2-2.

Figure 1–2 Web Server Acceleration



Benefits of Web Caching

Web caching provides the following benefits for Web sites:

- Performance

  Running on inexpensive hardware, Oracle Web Cache can increase the throughput of a Web site by several orders of magnitude. In addition, Oracle Web Cache significantly reduces response time to browser requests by storing documents in memory and by serving compressed versions of documents to browsers that support the GZIP encoding method.

- Scalability

  In addition to unparalleled throughput, Oracle Web Cache can sustain thousands of concurrent browser connections, meaning that visitors to a site see fewer application Web server errors, even during periods of peak load.

- High Availability

  Oracle Web Cache supports content-aware (Layer 4 - Layer 7) load balancing and failover detection. These features ensure that cache misses are directed to the most available, highest-performing Web server in the cluster. Moreover, a patent-pending capacity heuristic guarantees performance and provides surge protection when application Web server load increases.

- Cost Savings

  Better performance, scalability and availability translates into cost savings for Web site operators. Because fewer application Web servers are required to meet the challenges posed by traffic spikes and denial of service attacks, Oracle Web Cache offers a simple and inexpensive means of reducing a Web site's cost per request.

- Network Traffic Reduction

  Most requests are resolved by Oracle Web Cache, reducing traffic to the application Web servers. The cache also reduces traffic to backend databases located on computers other than the application Web server.

Features of Oracle Web Cache

The main features of Oracle Web Cache make it a perfect caching service for e-business Web sites that host online catalogs, news services, and portals. These features include:

- Static and Dynamically Generated Content Caching
- Cache Invalidation
- Performance Assurance
- Surge Protection of Application Web Servers
- Load Balancing of Application Web Servers
- Security Features
- Administration
- Compression

Static and Dynamically Generated Content Caching

Oracle Web Cache uses cacheability rules to store documents. These rules fall into two categories:

- Rules for static content, such as GIF, JPEG, or static HTML files
- Rules for dynamically generated content created using technologies like Java Server Pages (JSP), Active Server Pages (ASP), PL/SQL Server Pages (PSP), Java Servlets, and Common Gateway Interface (CGI). Support of these technologies enables Oracle Web Cache to recognize rules for the following:
  - Multiple-version documents for the same URL, that is, the same URL with slightly different content
  - Session-aware rules for pages containing session information
  - Personalization rules for pages containing personalized greetings, such as "Welcome <Name>", and session-encoded URLs

    See Also: "Caching Dynamically Generated Content" on page 2-5 for further information about dynamically-generated content

Cache Invalidation

Oracle Web Cache supports **invalidation** as a mechanism to keep its cache consistent with the content on the application Web servers, origin databases, or other dynamically generated means.

Administrators can invalidate cache content in one of two ways:

- Send an HTTP invalidation message to the computer running Oracle Web Cache

  When documents are invalidated and a browser requests them, Oracle Web Cache refreshes them with new content from the application Web server.

- Assign an expiration time limit to the documents

  When a document expires, Oracle Web Cache treats it like an invalid document, that is, if requested by a browser, it refreshes it with a updated content from the application Web server.

> **See Also:** "Cache Freshness and Performance Assurance" on page 2-2 for further information about invalidation

Performance Assurance

When a large number of documents have been invalidated, the retrieval of a new documents can result in overburdened application Web servers.

To handle performance issues while maintaining cache consistency, Oracle Web Cache uses built-in **performance assurance heuristics** that enable it to assign a queue order to documents. These heuristics determine which documents can be served stale and which documents must be refreshed immediately. Documents with a higher priority are refreshed first. Documents with a lower priority are refreshed at a later time.

The queue order of documents is based on the popularity of documents and the validity of documents assigned during invalidation. If the current load and capacity of the application Web server is not exceeded, the most popular and least valid documents are refreshed first.

> **See Also:** "Cache Freshness and Performance Assurance" on page 2-2 for further information about performance assurance

Surge Protection of Application Web Servers

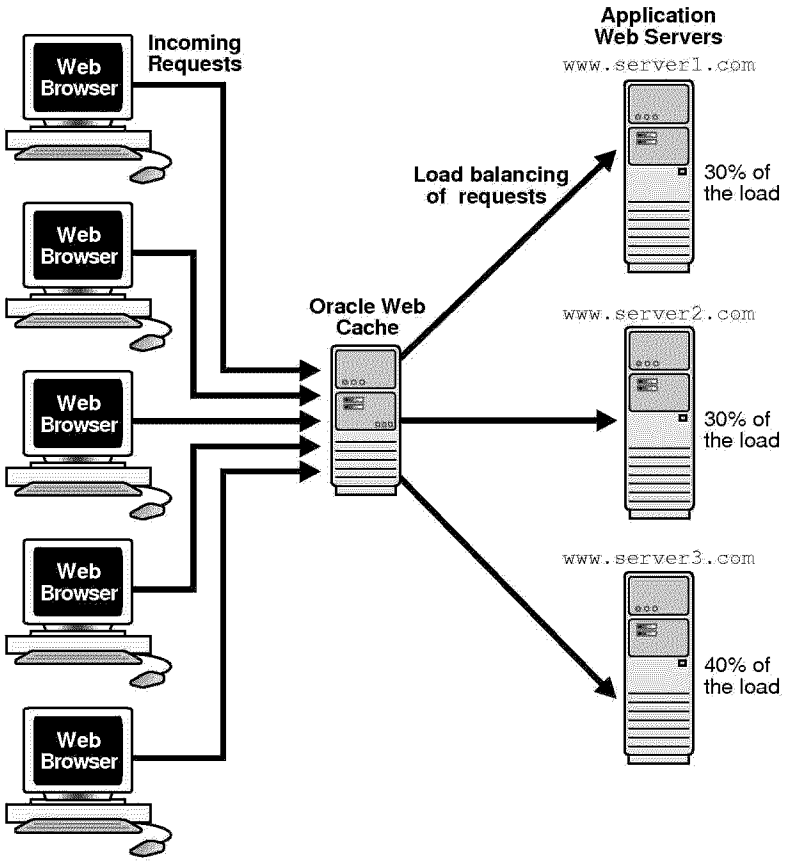
Oracle Web Cache passes requests for non-cacheable or stale documents to the application Web servers. To prevent an overload of requests on the application Web servers, Oracle Web Cache has a surge protection feature that enables you to set a limit on the number of concurrent requests that the application Web servers can handle. When the limit is reached, subsequent requests are queued to wait up to a maximum amount of time. If the maximum wait time is exceeded, Oracle Web Cache rejects the request and serves a site busy apology page to the Web browser that initiated the request.

Load Balancing of Application Web Servers

Most Web sites are served by multiple application Web servers running on multiple computers that share the load of HTTP requests. This feature enables Web sites to be built with a collection of servers for better scalability and reliability. Oracle Web Cache is designed to manage HTTP requests for up to 100 application Web servers. All requests that Oracle Web Cache cannot serve are passed to the application Web servers. Oracle Web Cache has a **load balancing** feature that distributes these requests over a set of application Web servers. To configure load balancing, you set the capacity (concurrent connections) for each application Web server. Based on the capacity assigned, Oracle Web Cache prescribes the relative percentage load of each application Web server.

Figure 1–3 shows three application Web servers, whereby 30 percent of traffic goes to `www.server1.com`, 30 percent goes to `www.server2.com`, and 40 percent goes to `www.server3.com`.

Figure 1–3 Load Balancing



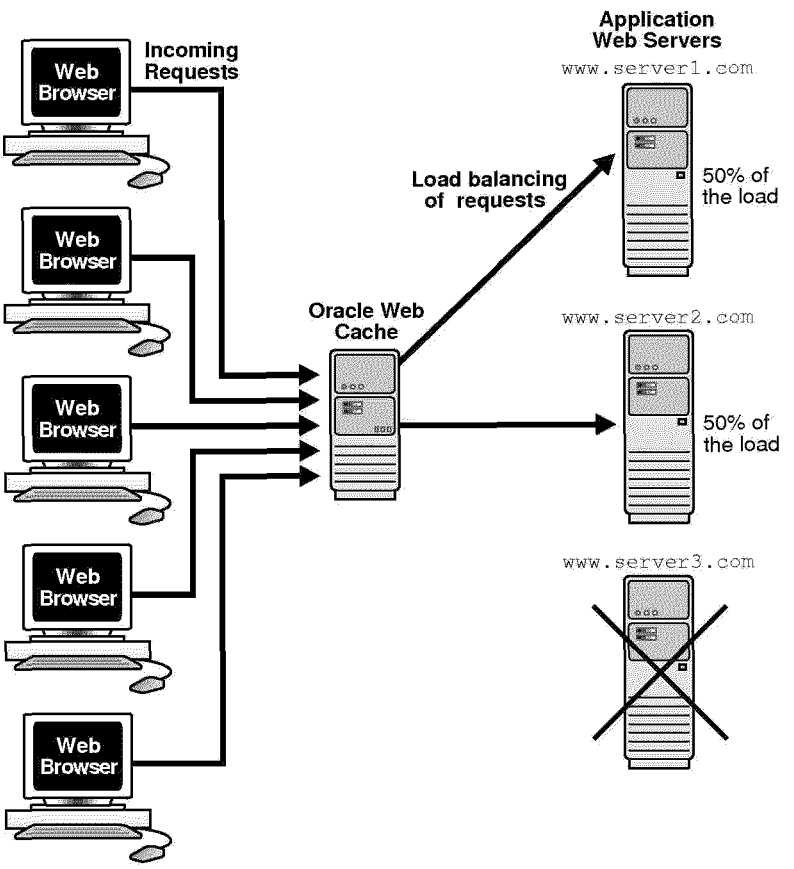
Backend Failover

After five continuous request failures, Oracle Web Cache considers an application Web server as failed. When an application Web server fails, Oracle Web Cache automatically distributes the load over the remaining application Web servers based on the on the remaining proportion. Oracle Web Cache polls the failed application Web server for its current up/down status every 60 seconds until it is back online.

The **failover** feature is shown in Figure 1–4. An outage of `www.server3.com`, which was receiving 40 percents of requests, results in 50 percent of the requests going to `www.server1.com` and 50 percent of requests going to `www.server2.com`. This is based on a 30 percent load for both `www.server1.com` and `www.server2.com`. If `www.server1.com` were to fail instead, based on a 30 percent load for `www.server2.com` and a 40 percent load for `www.server3.com`, 42.86 percent of the requests would go to `www.server2.com` and 57.14 percent of requests would go to `www.server3.com`.

When the failed server returns to operation, Oracle Web Cache will include it in the load mix as previously prescribed.

Figure 1–4 Failover



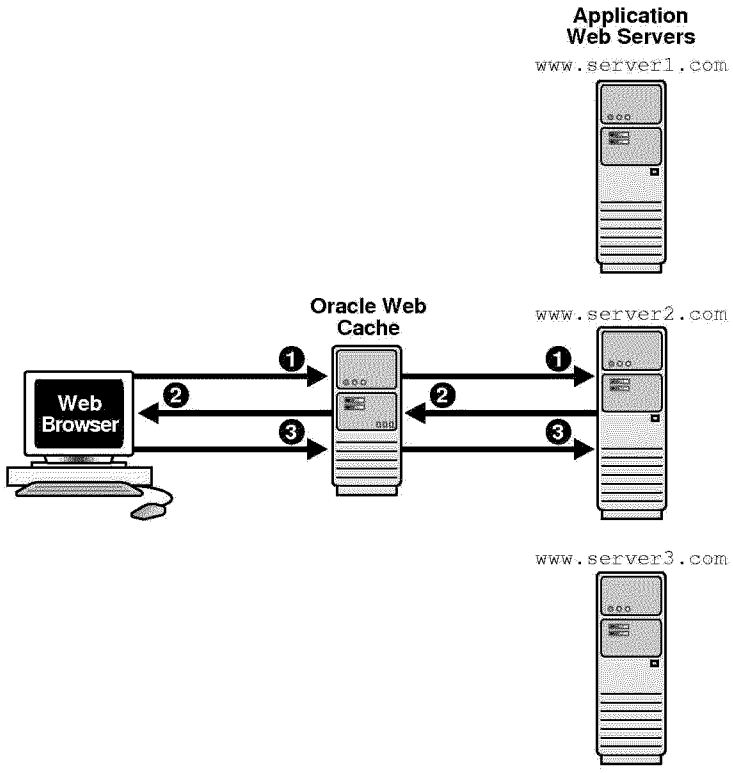
Application Web Server Binding

Oracle Web Cache supports Web sites that use session IDs and/or **cookies** to bind user sessions to a given application Web server in order to maintain state for a period of time. Web sites bind user sessions by including session data in the HTTP header or body it sends to Web browsers in such a way that the browser is forced to include it with its next request. This data is transferred either with parameters embedded in the URL or cookies, which are text strings stored on the client.

Figure 1–5 shows how Oracle Web Cache supports documents that use application Web server binding:

1. When a request first comes in, Oracle Web Cache uses load balancing to decide which application Web server to send it to. In this example, `www.server2.com` was chosen.
2. If the requested document requires application Web server binding, the application Web server sends the session information back to the browser through Oracle Web Cache in the form of a cookie or an embedded URL parameter.
3. Oracle Web Cache sends subsequent requests for the session to the application Web server that established the session, bypassing load balancing. In this example, `www.server2.com` handles the subsequent requests.

Figure 1–5 Application Web Server Binding



ORCL00107991

Security Features

Oracle Web Cache provides important security features including:

- Password authentication for administration and invalidation operations
- Control over which ports administration and invalidation operations can be requested from
- Timeout for inactive connections
- IP and subnet administration restrictions

Administration

Oracle Web Cache provides a graphical user interface tool called **Oracle Web Cache Manager** that combines configuration and monitoring options to provide an integrated environment for configuring and managing Oracle Web Cache and the Web sites it caches for. With the Oracle Web Cache Manager, you can easily:

- Configure Oracle Web Cache to cache for application Web servers
- Start and stop Oracle Web Cache
- Establish cacheability rules
- Monitor Oracle Web Cache and Web site performance
- Establish listening ports and security passwords

Compression

You can select to have documents compressed upon insertion into the cache. Oracle Web Cache is able to compress 300 KB files down to 3 KB, increasing the overall throughput by a factor of 100.

Exhibit 29

OIPE
NOV 24 2000
PATENT & TRADEMARK OFFICE

GP ~~2764~~ 2161

50277-0145

PATENT #

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:
Lawrence JACOBS, et al.

Serial No.: 08/962,536

Examiner: Yehdega Retta

Filing Date: October 31, 1997

Art Unit: 2764

RECEIVED
NOV 29 2000
Group 2100

For: METHOD AND APPARATUS FOR PERFORMING TRANSACTIONS IN A STATELESS WEB ENVIRONMENT WHICH SUPPORTS A DECLARATIVE PARADIGM

INFORMATION DISCLOSURE STATEMENT

Commissioner for Patents
Washington, DC 20231

Dear Sir:

This Information Disclosure Statement is being filed under Rule 37 CFR 1.97(i), wherein applicants are submitting references before the grant of a patent to be placed in the file but not considered by the Patent Office.

Accordingly, copies of the references as listed on the attached Form PTO 1449 are submitted herewith for placement in the file. No certification or fees are deemed necessary.

Respectfully submitted,
HICKMAN PALERMO TRUONG & BECKER LLP

Carl L. Brandt
Registration No. 44,555

1600 Willow Street
San Jose, California 95125-5106
Date: November 21, 2000
Telephone (408) 414-1080
Facsimile: (408) 414-1076

CERTIFICATE OF MAILING
I hereby certify that this correspondence is being deposited as first-class mail in an envelope addressed to the Commissioner for Patents, Washington, D.C. 20231:
on November 21, 2000 by

#32

SHEET 1 OF 1

INFORMATION DISCLOSURE CITATION IN AN APPLICATION

(PTO-1449)

OIPE JC47 NOV 24 2000 PATENT & TRADEMARK OFFICE

RECEIVED NOV 29 2000 Group 2100

ATTY. DOCKET NO.	SERIAL NO.
50277-0145	08/962,536

APPLICANT: Jacobs, et al.

FILING DATE	GROUP
October 31, 1997	2764

U.S. PATENT DOCUMENTS

EXAMINER'S INITIALS	PATENT NO.	DATE	NAME	CLASS	SUBCLASS	FILING DATE
YR	4,918,595	04/17/90	Kahn et al.	364	200	
	5,210,824	05/11/93	Putz et al.	395	145	
	5,212,793	05/18/93	Donica, et al.	395	700	
	5,249,290	09/28/93	Heizer	395	650	
	5,329,619	07/12/94	Page et al.	395	200	
	5,341,478	08/23/94	Travis, Jr. et al	395	200	
	5,592,654	01/07/97	Djakovic	395	500	
	5,752,246	05/12/98	Rogers et al.	707	10	
	5,761,507	06/02/98	Govett	395	684	
	5,778,224	07/07/98	Tobe et al.	395	670	
	5,802,291	09/1/98	Balick et al.	395	200.32	
	5,857,102	01/5/99	McChesney et al.	395	145	
	5,894,554	04/13/99	Lowery, et al.	395	200.33	
	6,070,191	05/30/00	Narendran, et al.	709	226	

FOREIGN PATENT DOCUMENTS

EXAMINER'S INITIALS	PATENT NO.	DATE	COUNTRY	CLASS	SUBCLASS	Translation Yes	Translation No
YR	553560A2	08/04/93	EPO	~			

OTHER ART (Including Author, Title, Date, Pertinent Pages, Etc.)

EXAMINER	DATE CONSIDERED
Yehdega Retta	2/2/01

EXAMINER: Initial if reference considered, whether or not citation is in conformance with MPEP 609; draw line through citation if not in conformance and not considered. Include copy of this form with next communication to Applicant.