

Oracle9i Application Server

*Scalability, Availability, and Load Balancing Options with
Oracle9iAS*

February 2001

Oracle9i Application Server.....	3
Scalability, Availability and Load Balancing.....	4
Scalability.....	4
Hardware Scalability	5
Data Scalability	5
Request Scalability.....	6
Application Scalability	6
Java Application Scalability in Oracle9iAS.....	8
Availability	9
No single point of failure	10
Session isolation	10
Connection rerouting.....	11
Death detection and restart	12
Failover	13
Load Balancing.....	13
HTTP Server with a Single Host	14
HTTP Server with Multiple Hosts.....	14
Apache JServ with Single and Multiple Hosts	15
Oracle EJE with Single Host.....	15
Oracle EJE with Multiple Hosts.....	17
HTTP Server and Oracle EJE.....	17
Middle-Tier Server Farms	18
Deployment Architectures.....	19
JSP/Servlet Application.....	20
One Host.....	20
Multiple Hosts	20
Multiple Hosts with Oracle9iAS Web Cache.....	21
Multiple Servers with Oracle9iAS Database Cache	22
PL/SQL Application.....	23
Summary	24

ORACLE9i APPLICATION SERVER

Oracle9i Application Server is Oracle's new application server that provides a simple, complete, and integrated platform for running Web sites and all types of Internet applications. Oracle9iAS provides support for open standards application development with CORBA, BC4J, and full support for the Java2 Enterprise Edition platform, database intensive programming with PL/SQL, and model-based development with Oracle9iAS Forms Services and Oracle9iAS Reports Services. Oracle9iAS combines the strength and reliability of mature Oracle technology with the power of new features such as the Oracle HTTP server powered by Apache, Oracle9iAS Web Cache, and Oracle9iAS Database Cache to dramatically improve the performance of your Web site. Oracle9iAS offers high levels of scalability, availability, and load balancing. Oracle9iAS can be deployed in a multitude of configurations, enabling you to re-deploy your applications for additional performance or reliability without needing to alter their application code.

Oracle9iAS provides the infrastructure needed to run all elements of your e-business and allows you to focus on your business operations and decision making:

- Oracle9iAS provides an integrated platform to build, deploy and maintain e-business Web sites using any standard technology and leveraging tight integration with the Oracle Database.
- Oracle9iAS allows you to aggregate all of your disparate Internet systems and Web content into personalized, secure portal pages for all of your users.
- Oracle9iAS enables you to run any Web site, portal or e-business application at least 3 times faster than any other application server.
- Oracle9iAS allows you to make your internet applications and Web sites accessible to traditional browsers and wireless devices;
- Oracle9iAS provides business intelligence solutions tightly integrated with the Oracle Database to allow you to make strategic decisions based on facts.
- Oracle9iAS allows you to manage your entire Web infrastructure within a comprehensive management framework.
- Oracle9iAS lets you connect your existing business systems and data stores to form an integrated e-business solution.

- Oracle9iAS lets you collaborate effectively with all lines of business within your organization in a scaleable and open standards manner.

This paper discusses how Oracle9iAS functions, and how it supports scalability, availability, and load balancing.

SCALABILITY, AVAILABILITY AND LOAD BALANCING

A key to the success of any Web site is how fast the server can deliver content to users. If a Web server takes too long to respond, or fails, users will take their business elsewhere. Three areas that determine how well your Web site performs are scalability, availability, and the ability to balance incoming loads across multiple servers. Oracle9iAS provides a highly efficient platform for Web site deployment that performs very well in all three areas:

- **Scalability:** Quality that indicates how well your Web site can respond as the user demands increase. Oracle9iAS provides hardware, data, request, and application scalability. To provide hardware scalability, Oracle9iAS is available on all the major hardware platforms so you can choose different capacity servers without redeveloping your hosted applications; to provide data scalability, Oracle9iAS Database Cache serves database queries from the middle tier without having to route the request to your back-end database; to provide request scalability, Oracle9iAS Web Cache caches and serves commonly requested content without routing the request to your Web server; to provide application scalability, Oracle Enterprise Java Engine (EJE) offers a unique session based architecture
- **Availability:** Quality that indicates how your Web site responds in the case of a software or hardware failure. Oracle9iAS has no single point of failure and can be deployed in a fully redundant configuration. Oracle9iAS uses session isolation to isolate executing user sessions from each other so single application failures have minimal impact on other users. Oracle9iAS automatically detects system components failures, re routes connections, restarts failed processes, and can move executing application code to a different node in some cases.
- **Load Balancing:** Feature that allows you to share the distribution of requests amongst different hosts operating together as a virtual server. Oracle9iAS is completely flexible and can be deployed in many different configurations. Oracle9iAS provides load balancing mechanisms and can operate fully with third party load balancing products such as Cisco Local Director.

Scalability

The *scalability* of a system generally refers to the ability of the system to provide satisfactory performance under large load. Scalability is limited by any bottleneck in the system, which could be the memory of a hardware node, the node's processing power, or other system limitations not restricted to hardware. In the first two cases, the system is declared as memory-bound or CPU-bound, respectively. The

most effective way to increase system scalability is to identify its bottleneck and reconfigure the system to eliminate it. Oracle9iAS provides a scalable environment by enabling addressing a variety of potential system bottlenecks in four primary ways:

- Oracle9iAS runs on a broad set of hardware and operating systems, so administrators can upgrade their hardware without modifying their applications.
- Oracle9iAS boosts system scalability by caching database data and stored procedures on the middle tier. This allows the back end database to serve increased numbers of concurrent users with the same computing resources.
- Oracle9iAS boosts Web server performance and scalability of Web servers with intelligent caching of commonly accessed HTML pages. This dramatically increases site scalability and performance, without the need for additional computing resources in the middle tier.
- Oracle9iAS can be deployed on single node or multi-node clusters to scale both stateless and stateful applications.

Hardware Scalability

The Oracle Internet Platform, comprising both Oracle9iAS and the Oracle database, is available on a large selection of hardware and operating systems, scaling from low-end hardware to high-end clusters. Oracle9iAS and the Oracle Database support Microsoft operating systems, all major Unix platforms, and a variety of other systems. Therefore administrators can upgrade hardware and operating systems without changing their software platform or rewriting their applications.

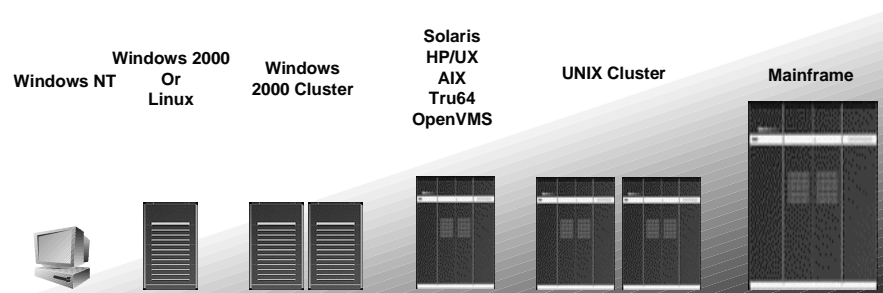


Figure 1: Hardware Scalability - Oracle9iAS and the Oracle Database run on a broad selection of hardware and operating systems.

Data Scalability

In systems that run data-intensive applications, a single-node database instance may often be the scalability bottleneck. One possible problem is that the database node may be memory constrained and frequent disk paging impairs the

performance. This problem can often be solved inexpensively by simply adding more memory to the server.

A more common problem is that the database node is CPU-bound. Many systems use a single database node to perform all the query and transaction processing for substantial concurrent client populations. Application code implemented with Oracle database stored procedures perform better when it runs close to database data, but these applications take up additional CPU cycles on the database node. The database CPU constraint becomes more pronounced and evident as companies start to leverage their data by making database applications available to many users over their corporate Intranet and the Internet. Oracle9iAS with the Oracle9iAS Database Cache addresses this problem directly by offloading data and application processing from the back end database node onto the middle tier.

When Oracle9iAS Database Cache is used, the back end database can support more users because it will only service data updates and fewer data queries. In addition, applications derive better performance by querying a local data cache instead of a database over a network.

Request Scalability

Oracle9iAS increases the request handling capacity of your existing infrastructure with Oracle9iAS Web Cache. Oracle9iAS Web Cache resides in front of the Web server processes and stores frequently requested pages directly in memory. Oracle9iAS Web Cache then intercepts requests destined for the Web servers and serves the requested content directly from memory, bypassing the need for the Web server to respond to each and every request for content. The Web servers only see requests for new content and for content that is determined to be non cacheable. Using Oracle9iAS Web Cache results in a dramatic increase in the request handling capacity of Web sites.

Application Scalability

Oracle9iAS provides a very scalable infrastructure for executing applications. This section discusses how the Oracle9iAS environment provides scalability for both stateless and stateful applications.

Stateless and Stateful Scalability

Oracle9iAS supports both stateless and stateful transactional Web applications. A *stateless* application maintains no state information within its environment. However, it may maintain state information in a persistent store such as a database or a cookie in the browser. An example of a stateless application is a shopping cart program that stores the contents of a user's cart in a database table. Every time the user makes a request, the application retrieves the state information from the persistent store, processes the request, updates the database, and sends a response to the client. The application itself does not keep track of the user's shopping cart

between successive client calls. This type of application may be implemented with CGI scripts, stateless Java servlets, or in a variety of other ways.

Figure 1 illustrates how requests for stateless applications are serviced by different middle tier nodes.

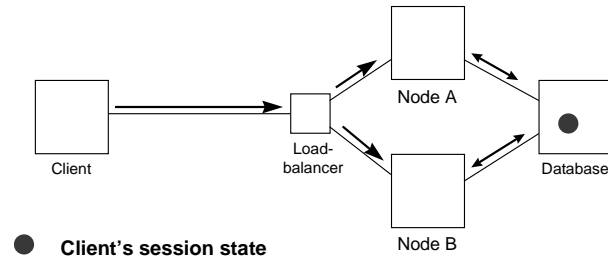


Figure 2: Stateless application. No client-specific session state is maintained within the application memory on Node A. The client session state is stored in the database.

On the other hand, a *stateful* application does maintain session state information within its runtime environment between successive client calls. The shopping cart application may alternatively be implemented using this model. In this case, the application keeps track of the contents of a user's shopping cart instead of storing this information every time in the database. Java servlets, for example, maintain session state by attaching state information to an HttpSession object that is specific to each user session. Every time a user issues a new HTTP request to the server, the server associates the request with the HttpSession object for that user.

Figure 3 illustrates a stateful application, where a client's session state is maintained in a specific application instance, running on a specific node.

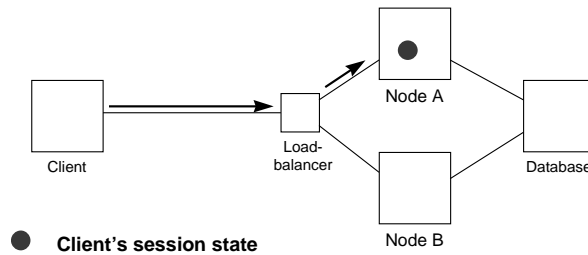


Figure 3: Stateful application. Client specific session state is maintained within Node A. On every subsequent request, the client must be serviced by Node A (and not Node B) so that the session state may be accessed.

Many applications are written to use either a stateful or stateless model, and there are various benefits of each approach. A large percentage of Web applications running on the Internet are implemented using the stateless model. As demonstrated in Figure 4 above, client requests for stateless applications can be served by any of multiple application server instances. This can produce a more scalable and fault-tolerant architecture. Stateful applications, on the other hand, are

usually easier to write because the application needs to do less state maintenance. A stateful implementation can perform better sometimes, for example, in applications whose client sessions consist of a substantial amount of state information and where many calls are made to the server in a short period of time. The stateful approach avoids repeatedly loading session state on each call into the application environment.

Oracle9iAS provides scalability for both stateless and stateful applications. For Java applications, the following provides guidelines as to where you should run your Java within Oracle9iAS.

- Stateless Java applications scale well in the JDK JVM.
- Stateful Java applications scale well in Oracle EJE.

Both JVMs support both stateless and stateful applications, but their individual architectures are more suited towards one or the other, as explained in the following section.

Oracle9iAS also provides a highly scalable infrastructure for Oracle9iAS Forms, Oracle9iAS Reports, and PL/SQL applications. See the component specific white papers for more detail on scalability characteristics of these Oracle9iAS components.

Java Application Scalability in Oracle9iAS

The JDK JVM scales by giving quick performance to many clients. This works well for stateless (or lightly stateful) Java applications because the VM does not get weighted down by holding onto a lot of state.

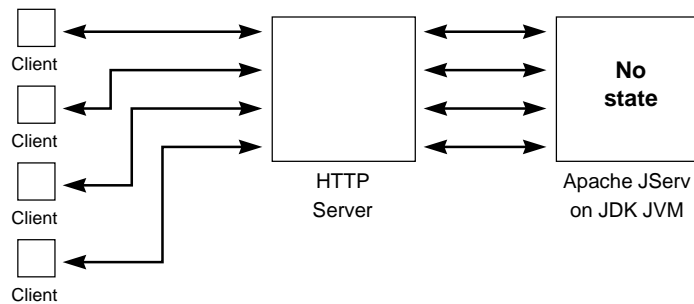


Figure4: Stateless application in JDK JVM. Additional clients do not contribute additional state to be managed between successive requests. The JDK JVM performs well in this scenario.

However, *stateful* applications force the JDK JVM to perform a lot of concurrent memory management when multiple users access the system. Managing state may inhibit the scalability of the JDK JVM.

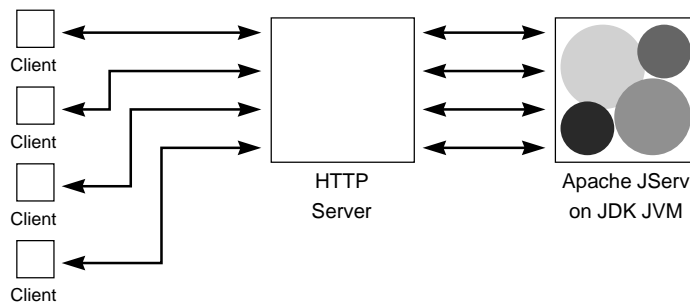


Figure 5: Stateful application in JDK JVM. Managing the memory may slow down execution of this application

Oracle EJE is a session-based JVM that handles stateful applications exceedingly well. Performance does not degrade until the capacity of the hardware is reached. As Oracle EJE segregates clients memory spaces, the JVM can garbage collect each user's memory space independently. This architecture avoids *concurrent garbage collection*, which often constitutes the major scalability bottleneck in running heavily stateful applications on a typical JVM.

Figure 6 illustrates the virtual JVM approach Oracle EJE uses. Each client appears to have their own dedicated JVM.

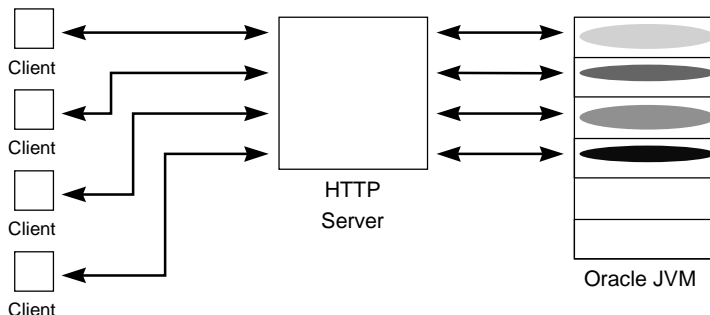


Figure 6: Stateful applications in Oracle EJE. Concurrent garbage collection does not occur in Oracle EJE due to its use of segregated memory spaces.

Availability

Users running applications on Oracle9iAS should perceive little or no loss of service during many types of hardware and software outages. Oracle9iAS provides a number of features and mechanisms designed to keep your system available despite limited server failures:

- Oracle9iAS has no single point of failure.
- Oracle9iAS isolates sessions to minimize impact of session outage.
- Oracle9iAS can automatically detect failure, reroute connections, restart processes, and in some cases it performs session failover.

No single point of failure

Oracle9*i*AS has a flexible deployment model so you can deploy an architecture that exposes no single point of failure. This means that despite the failure of any node in the system, Oracle9*i*AS will continue to function and service client requests. Figure 7 illustrates an example deployment of Oracle9*i*AS and the Oracle database, which has no single point of failure in the entire system, including both the application server and the database. Any node can fail and the system will continue to function. The load-balancer can send requests to any of multiple Oracle HTTP Servers. The Oracle HTTP Servers, in turn, can dispatch requests to any Apache JServ instance. The OCI programs running in Apache JServ will query any of the Oracle9*i*AS Database Cache instances, which are supported in the back end by multiple instances of the Oracle Database running Oracle Parallel Server.

Please note that the network load balancer in Figure 7 may be a third party product such as Cisco Local Director, which itself may be deployed in a redundant configuration.

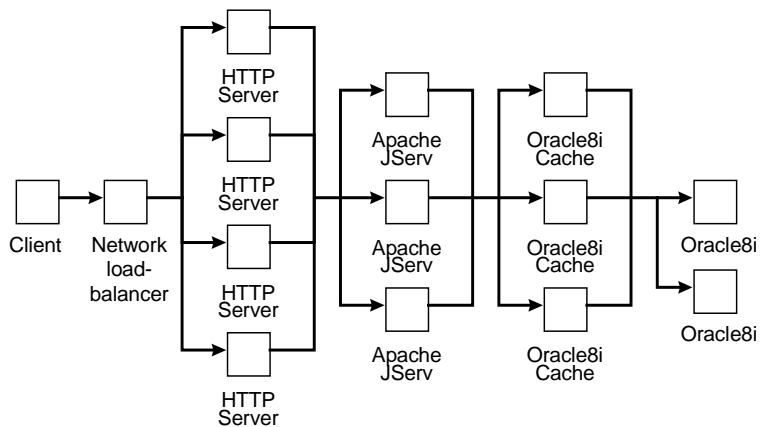


Figure 7: No single point of failure. An example deployment of Oracle9*i*AS and the Oracle Database which has no single point of failure.

Session isolation

Session isolation is a property of some architecture models that protects users sessions from each other to minimize the extent of damage in case of failure. The failure of one user session does not affect other user sessions. Oracle9*i*AS employs session isolation in Oracle EJE, Oracle PL/SQL, Oracle9*i*AS Forms and Oracle9*i*AS Reports.

Session isolation is a key differentiating feature between Oracle EJE and the JDK JVM. As illustrated in Figure 8, the JDK JVM does not employ a session isolating mechanism. If one client encounters a problem that causes the JDK JVM to fail, all users with sessions on that JDK JVM instance will be affected.

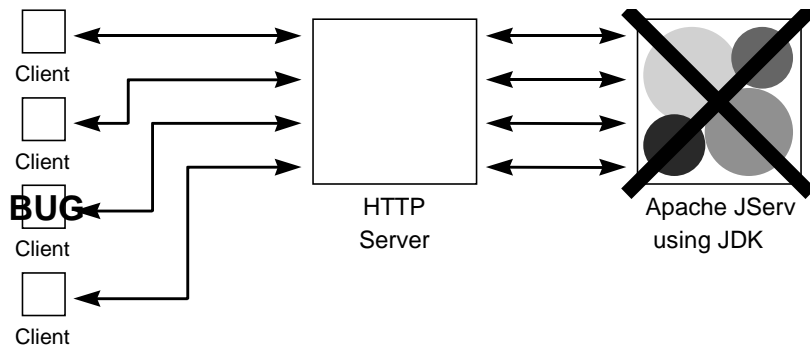


Figure 8: JDK JVM. There is no session isolation in a typical JDK JVM. The failure of one user session may cause all user sessions to fail.

Figure 9 demonstrates how session isolation protects concurrent users in Oracle EJE. Because Oracle EJE uses a multi-process shared server architecture it effectively insulates concurrent sessions from each other so that the worst an erroneous Java session can do is cause the failure of its executing process. Only the single client's state will be lost. No other user sessions are affected, and the server recreates the downed process for the affected client.

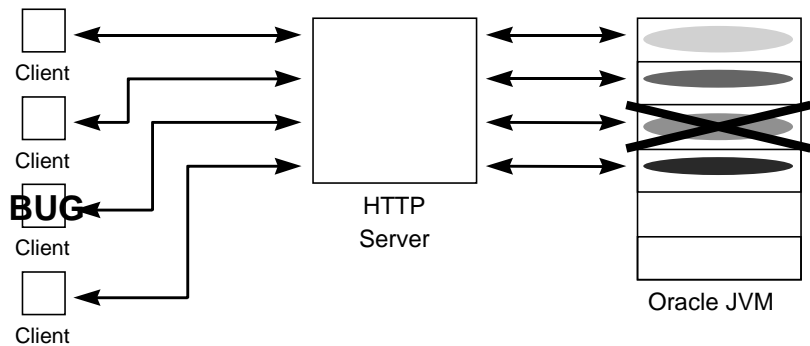


Figure 9: Oracle EJE isolates sessions from one another. In the case that one session fails, the others sessions are unaffected.

Connection rerouting

What happens to an executing session in Oracle9iAS when the process or node servicing its request suddenly fails? In some scenarios, client requests to the session can be transparently rerouted to alternate servers. When an Oracle9iAS stateless application fails, client requests are rerouted to alternate instances of the application. If a client accesses a stateful application, the client may be rerouted to the instance of the application through an alternate route.

Figure 10 illustrates an example of connection rerouting in the case of a failed HTTP Server node. In the example, the client is running a stateful application in Oracle EJE. The client will continue accessing its session because the server transparently reroutes requests through functional services.

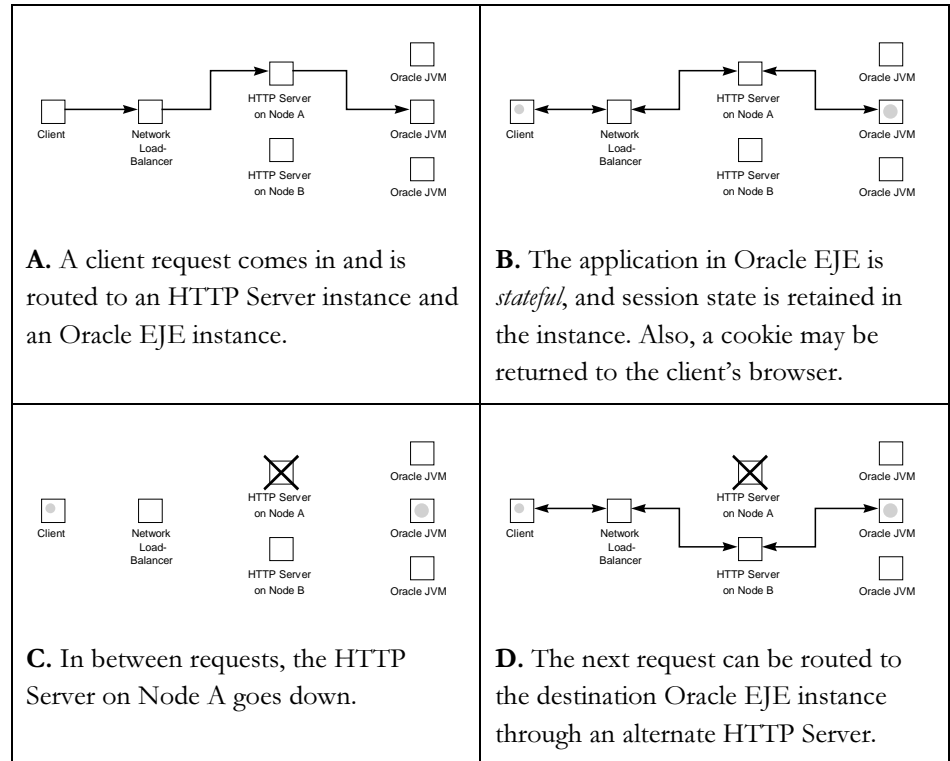


Figure 10: Connection rerouting in the case of a failed Oracle HTTP Server node.

Note that if only an HTTP Server process fails and not the entire node, then Oracle9iAS detects this failure and restarts the process.

Death detection and restart

If a server process fails, the system should take appropriate action, cleaning up memory and restarting the failed processes. Oracle9iAS detects the failure of the following types of processes and restarts them as necessary:

- **HTTP Server:** The *Watchdog* process in the HTTP Server monitors HTTP Server child processes and restarts a failed process.
- **Apache JServ:** `mod_jserv`, which runs within each HTTP Server process, detects the death of any Apache JServ instance and ceases routing requests to the instance. Oracle9iAS will support the automatic restart of remote Apache JServ instances in a future release.
- **Oracle EJE, Oracle PL/SQL, Oracle Database Cache:** The *Process Monitor (PMON)* process manages the server processes of these Oracle9iAS services. It detects the death of a server process and restarts it after cleanup.

Failover

Despite various potential failures (e.g. someone pulls the plug on a computer), clients should not perceive a loss of service. Failover is the infrastructure to hide system failure from users. Following failure of *stateless* services, Oracle9*i*AS will route requests to alternate instances of the service. This is similar to connection re-routing. After failure of *stateful* services, Oracle9*i*AS can re-route some types of session state to alternate instances. For example, Oracle9*i*AS offers support for stateful failover of session state in Oracle9*i*AS Database Cache.

Figure 11 illustrates the transparent application failover (TAF) functionality of Oracle9*i*AS Database Cache. If a cache node goes down, client sessions will be recreated automatically on an alternate instance.

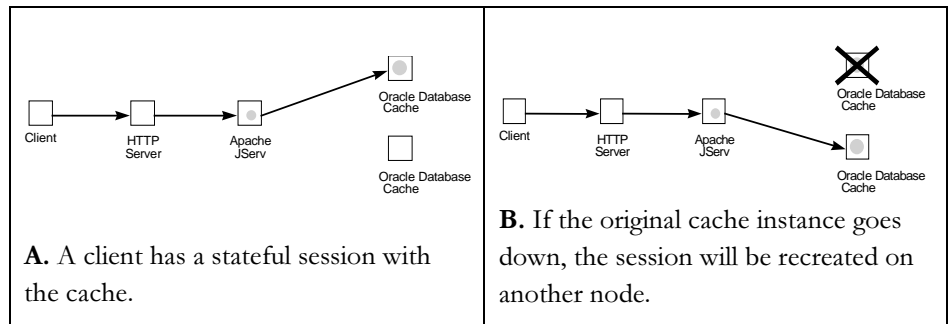


Figure 11: Stateful session failover with Oracle9*i*AS Database Cache.

Load Balancing

Effective load balancing helps maximize scalability because a system can make efficient use of its processing resources. Oracle9*i*AS load balances efficiently both between threads and processes on a single node, and between nodes in a multi-node deployment. Further, Oracle9*i*AS can be deployed in middle tier server farms.

This section examines the load balancing functionality of Oracle9*i*AS components of following example deployments:

- HTTP Server with single host
- HTTP Server with multiple hosts
- Apache JServ with single host and multiple hosts
- Oracle EJE with single host
- Oracle EJE with multiple hosts
- HTTP Server and Oracle EJE
- Middle tier server farms

Many of the other Oracle9iAS services exercise load balancing techniques as well. Please see the separate white papers for details on the load balancing capabilities of the Oracle9iAS Forms, Oracle9iAS Reports Server, and other component services.

HTTP Server with a Single Host

The Oracle HTTP Server uses a simple but efficient mechanism to load balance between HTTP Server processes within a single instance of the service. The master HTTP Server process does not service client requests itself, but spawns and monitors a group of child processes. The child processes take turns accepting HTTP requests from a shared socket by using a *mutex* (a mutually exclusive lock that can only be acquired by one entity at a time.) There is a single mutex instance, and only the child who currently owns the mutex is allowed to pull a request from the socket. Once a child receives a request, but before it begins servicing the request, it releases the mutex which can then be acquired by another child. In this way, access to the socket is serialized, but children may service requests in parallel. Figure 12 shows this mechanism.

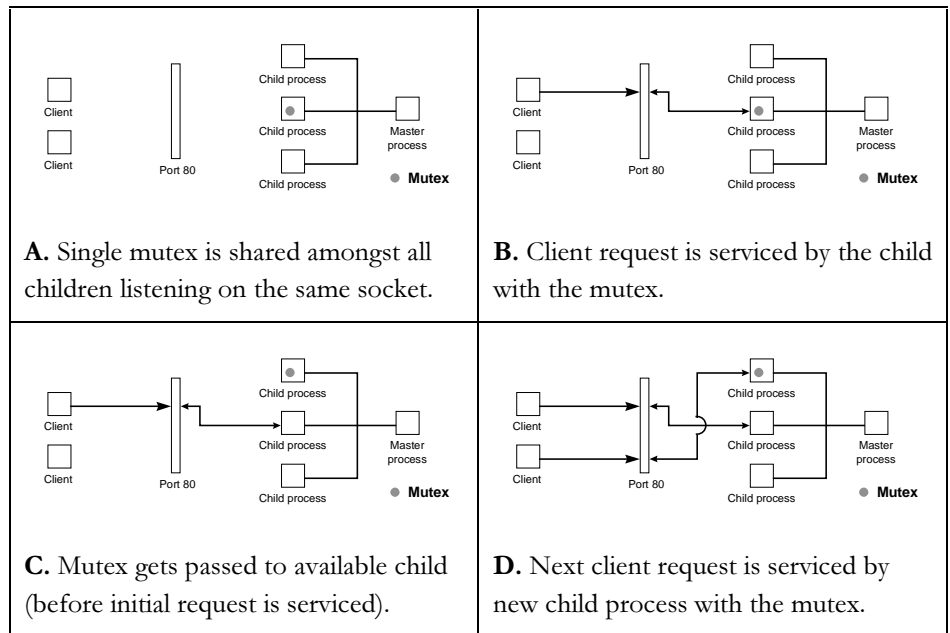


Figure 12: Load balancing in the Oracle HTTP Server on a single host.

HTTP Server with Multiple Hosts

Oracle HTTP servers may be run on multiple nodes. Client requests can be load balanced over the separate host instances using a variety of techniques: DNS round-robin, dedicated third party hardware and software mechanisms such as Cisco Local Director, or by utilizing the load balancing capabilities of Oracle9iAS Web Cache.

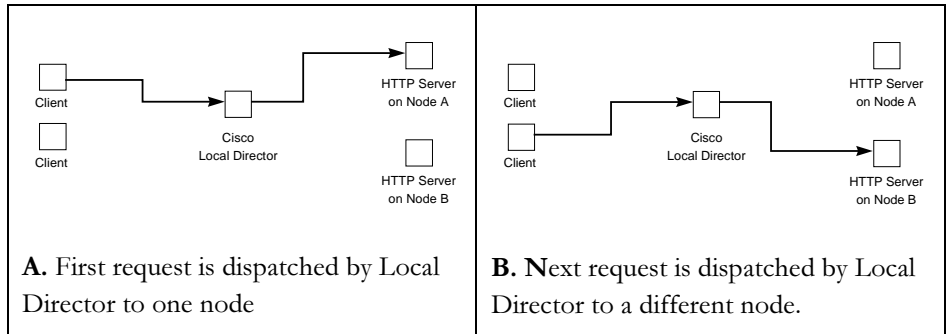


Figure 13: Load balancing Oracle HTTP Servers on multiple hosts using Cisco Local Director.

Apache JServ with Single and Multiple Hosts

The Oracle HTTP server (via `mod_jserv`) load-balances servlet requests to Apache JServ instances. Apache JServ instances can run concurrently on a single host or be distributed over multiple hosts. `mod_jserv` allocates new requests to the servlet containers based on a weighted algorithm where the system administrator provides weights for the various container instances. In this way, Apache JServ instances running on more powerful hardware can be allocated more requests than instances on less powerful machines. Figure 14 illustrates an example of this operation.

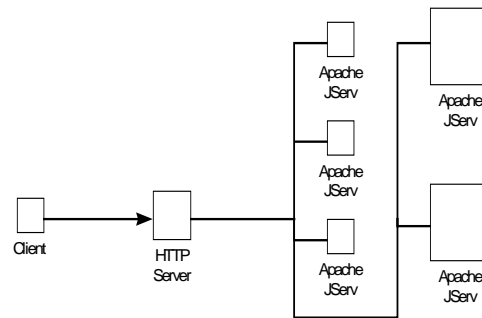


Figure 14: Load balancing Apache JServ. Apache JServ can be instantiated multiple times on a single node, or distributed across many nodes.

Oracle EJE with Single Host

Oracle EJE, Oracle PL/SQL Engine, and Oracle Database Cache utilize the Oracle Database Multi-Threaded Server, a sophisticated load balancing mechanism that maximizes throughput by making efficient use of server processes. This mechanism consists of two main parts. First, a client request can be load balanced over the server's *dispatcher* processes. Second, the dispatcher processes place requests in a common queue, which effectively load balances the request processing over multiple *shared server* processes. This mechanism is illustrated in Figure 15 and Figure 16. The figures reference the Oracle EJE as an example, but could also be discussing the Oracle PL/SQL Engine or Oracle Database Cache.

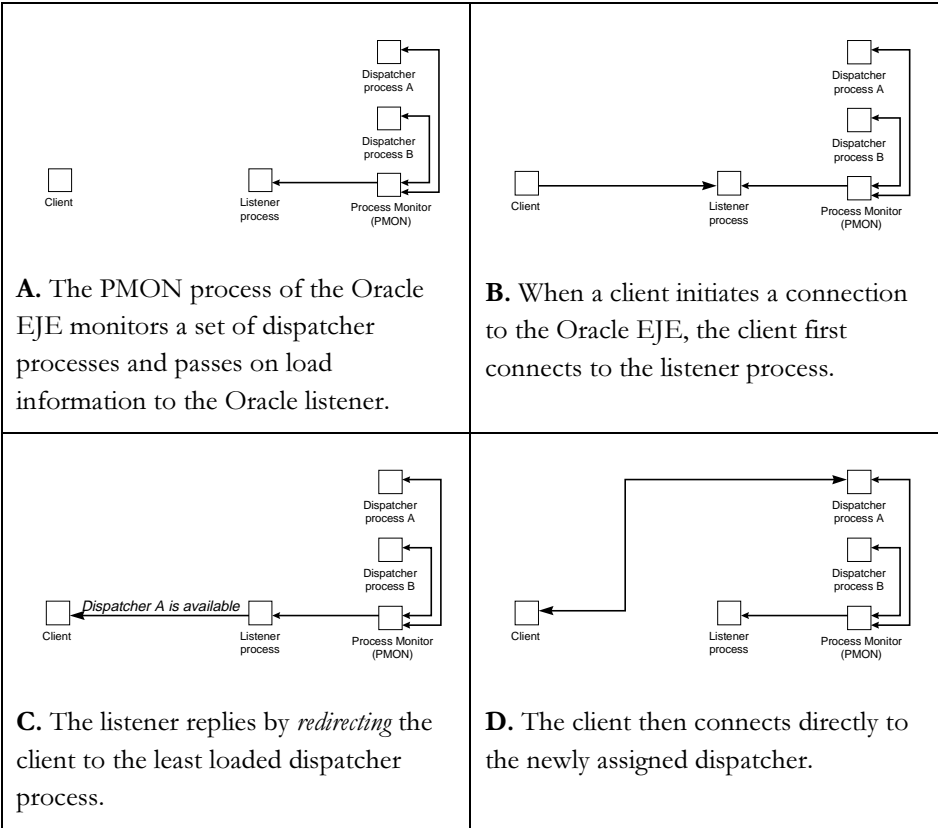
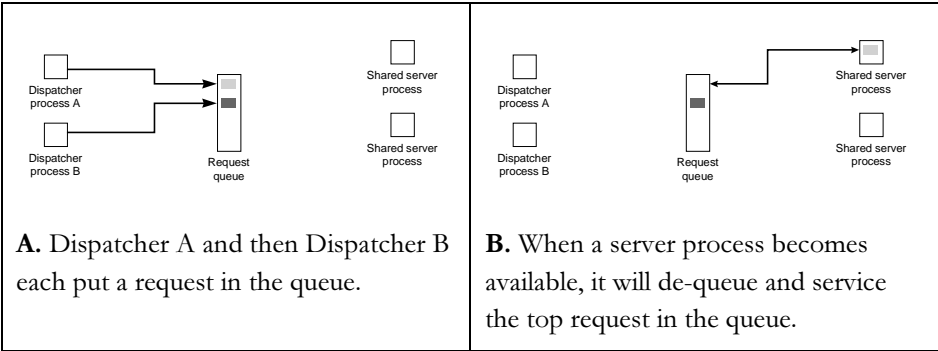


Figure 15: Load balancing over dispatchers in Oracle EJE.

Oracle EJE not only load balances over dispatchers, it performs further load balancing over the shared server processes that actually execute the request. The dispatcher processes put requests into a shared queue. Server processes de-queue and service requests on a first-come, first-served basis. Figure 16 illustrates load balancing over shared processes.



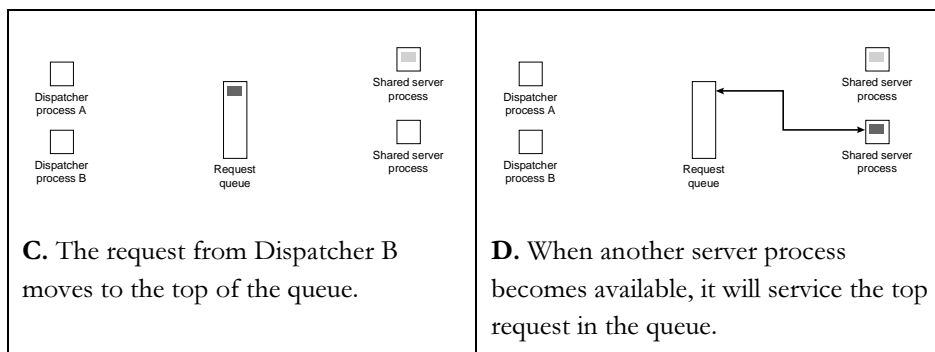


Figure 16: Load balancing over shared server processes in the Oracle EJE.

Oracle EJE with Multiple Hosts

Oracle EJE can also be load-balanced over multiple nodes. In addition to rerouting client requests to dispatcher processes on a single node, the listener uses a simple algorithm to load-balance requests to dispatchers distributed over multiple nodes. The listener selects a dispatcher by first choosing the least loaded node, and then selecting the least loaded dispatcher on that node. Figure 17 illustrates load balancing client requests over multiple nodes.

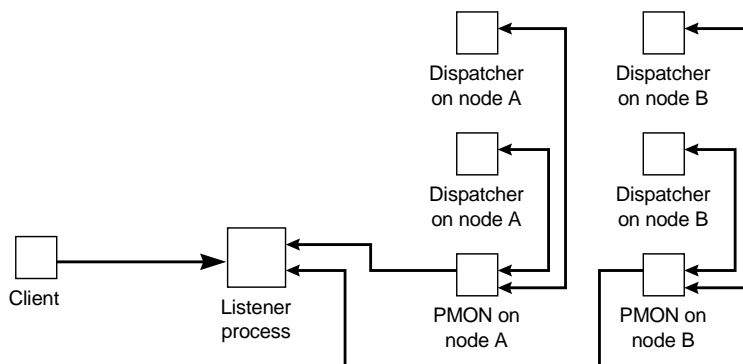


Figure 17: Load balancing the Oracle EJE over multiple nodes. The Oracle EJE listener process redirects a client request to the least-loaded dispatcher on the least-loaded node.

HTTP Server and Oracle EJE

Figure 18 puts the two pieces together, illustrating the five levels of load balancing that Oracle9iAS provides to a client HTTP request that is serviced with Oracle EJE. The circles in the figure labeled 1 through 5, depict the levels of load balancing provided by Oracle9iAS ,

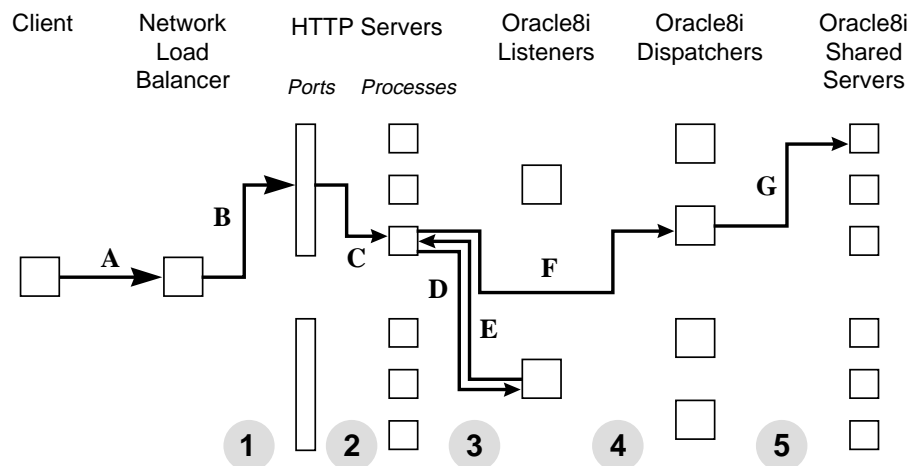


Figure 18: Five levels of load balancing are involved in the servicing of a client request that utilizes the Oracle HTTP Server and Oracle EJE.

- A.** A client HTTP request comes in through the network load balancer
- B.** The first level of load balancing occurs as the request is routed to one of the HTTP server nodes. The network load balancer selects a node randomly, or it may choose the least loaded node.
- C.** Within the HTTP server node, the server processes take turns servicing requests. A ready process services the request.
- D.** Oracle EJE can load-balance over multiple listener processes. A listener is chosen at random.
- E.** The listener determines which Oracle EJE node has the lightest load, and then which dispatcher on that node is most available. The listener redirects its client (the HTTP Server process) to the selected dispatcher.
- F.** The HTTP server process connects directly to the assigned Oracle EJE Database dispatcher.
- G.** The dispatcher enqueues the request, which is serviced by a shared server process. When a shared server process becomes available, it services the request.

Middle-Tier Server Farms

Middle-Tier server farms generally refer to a popular deployment architecture in which a company runs their middle tier application servers on a large set of inexpensive hardware, such as 1 or 2 CPU Intel machines. They often run similar software configurations on each node, and when additional scalability is required, they simply purchase more hardware and replicate the software environment. A network load balancer distributes requests over the nodes of the farm. Oracle9iAS can readily be deployed in middle-tier server farms, as illustrated in Figure 19.

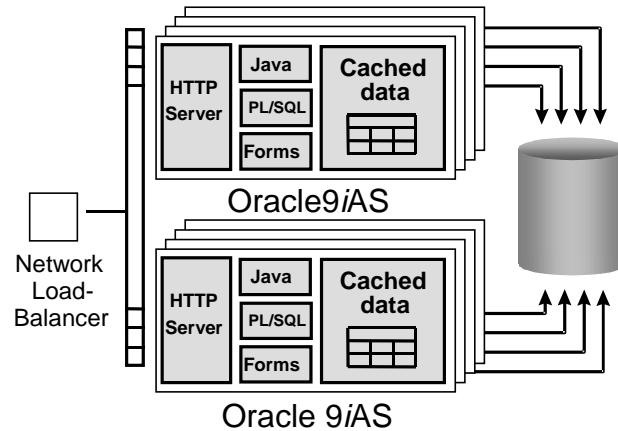


Figure 19: Oracle9iAS deployed on a middle tier server farm. The figure illustrates an Oracle9iAS configuration utilizing the Oracle HTTP Server, Java, PL/SQL, Oracle9iAS Forms, and the Oracle9iAS Database Cache.

DEPLOYMENT ARCHITECTURES

Oracle9iAS has a flexible deployment model so its services can be deployed on single nodes or on distributed systems in innumerable configurations. These configurations can be grouped into three basic categories:

- **One Host.** This is the simplest configuration, which hosts both the database and all Oracle9iAS components on a single machine. The one-box configuration is good for development purposes and may be adequate for small application deployments.
- **Multiple Hosts.** The multi-tier configuration places components of Oracle9iAS onto separate tiers from the database. Such configurations usually offer better application scalability and increased availability from the Oracle HTTP Server and other Oracle9iAS components.
- **Multiple Hosts with Oracle9iAS Web Cache.** This configuration uses Oracle9iAS Web Cache as a server accelerator to offload processing occurring on the middle tier. Oracle9iAS Web Cache's load balancing facilities can be used to distribute requests over multiple Oracle HTTP Server instances.
- **Multiple Hosts with Oracle9iAS Database Cache.** This configuration adds Oracle9iAS Database Cache, increasing application *and* data scalability.

This section applies these three basic deployment categories to two example application models. Specifically, it examines a JSP/servlet application, and then a PL/SQL application. Each example illustrates deployment in the one-host, multiple hosts, and multiple hosts with Oracle9iAS Database Cache configurations.

This discussion does not focus on application implementation because the Oracle9iAS model enables administrators to re-deploy functioning applications to

address new requirements for scalability, reliability, and availability without changing the application code.

JSP/Servlet Application

This section examines the basic deployment options for a typical JSP or Java servlet application. The example assumes that the Java application will be run in Apache JServ on the JDK JVM, although using the Oracle JVM may also be a viable option.

One Host

The most basic configuration for a general JSP or servlet application is illustrated in Figure 20. Note that in this and other figures below, the outer rectangles represent hardware nodes. Software services and elements are depicted in various shapes within the node. In this example, the Oracle HTTP Server, Apache JServ, and an Oracle database instance are co-resident on a single machine.

The one-box configuration is simple, inexpensive, and suitable for development and small deployments. Production systems will often be better served by a multi-tier configuration.

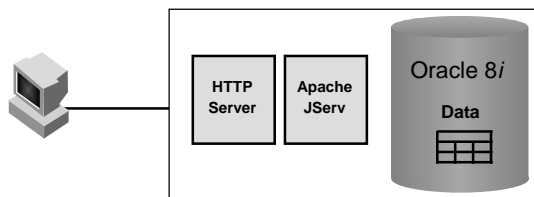


Figure 20: JSP/servlet application with one-host. This simple and inexpensive configuration places all required services on a single node.

Multiple Hosts

The multi-tier hosts configuration places the Oracle HTTP Server and Apache JServ services on a separate server than that of the database. In the multi-tier deployment example illustrated in Figure 21, these two Oracle9iAS services are running together on each middle tier node. Note the stacked rectangles in the figure indicate that multiple machines can be used on the given tier for increased scalability and reliability. The dotted line between the tiers represents a firewall, and is included in the figure simply as a suggestion. In this and in many cases below, there are multiple choices for where to deploy a firewall. In this example, for instance, the firewall could alternatively be placed outside (to the left) of the Oracle9iAS node.

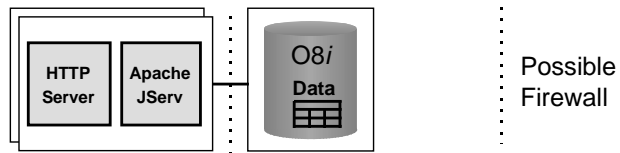


Figure 21: JSP/Servlet application and Oracle HTTP Server tiers. This configuration puts the Oracle HTTP Server and Apache JServ on a separate node from the database instance. This will generally be more scalable and fault-tolerant than the one-host configuration.

Figure 22 presents a variation on the preceding configuration. It separates the Oracle HTTP Server and Apache JServ to each run on independent nodes. This configuration produces a more reliable servlet environment as the servlet containers each have their own node. Note that any of the Oracle HTTP Server nodes can dispatch requests to any of the Apache JServ nodes. This is represented by the three intersecting lines connecting the two tiers in the figure.

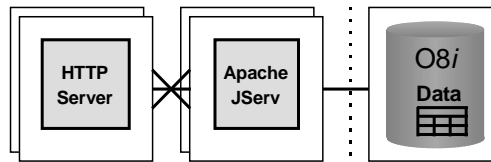


Figure 22: JSP/servlet application with separate Oracle HTTP Server and the Apache JServ tiers. This configuration separates Oracle HTTP Server and the Apache JServ servlet container(s) such that each runs on independent nodes.

Multiple Hosts with Oracle9iAS Web Cache

Oracle9iAS Web Cache is positioned in front of one or more Web servers to cache content generated by those servers. Oracle9iAS Web Cache then delivers that content to Web browsers. When Web browsers access the Web site, they send HTTP requests to Oracle9iAS Web Cache which acts as a virtual server for the Web site, masking the existence of the Web server farm and the database. If the requested content has changed, Oracle9iAS Web Cache retrieves the new content from the Web servers according to the relative load on each server.

Oracle9iAS Web Cache can be deployed on the same node as the origin Web server or on a dedicated node of its own. Figure 23 demonstrates how Oracle9iAS Web Cache may be co-located with the Web server(s) on the same machine.

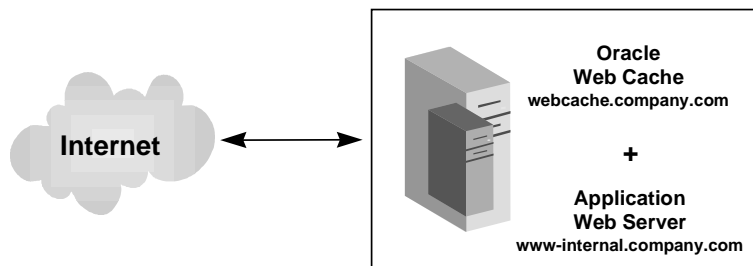


Figure 23: Oracle9iAS Web Cache on same node as Web server

In a cluster scenario, a network load balancer distributes requests across the cache instances running on each node in the server farm. Each Oracle9iAS Web Cache instance is typically configured with the host name of the Web server(s) running on its own node, and interprocess communication (IPC) is used to pass requests between the cache and the Web server(s). Because Oracle Web Cache consumes memory and CPU, co-location is only viable if the cache and the Web server(s) do not contend for resources.

Oracle Web Cache may also be deployed on a dedicated node, as illustrated in Figure 24.

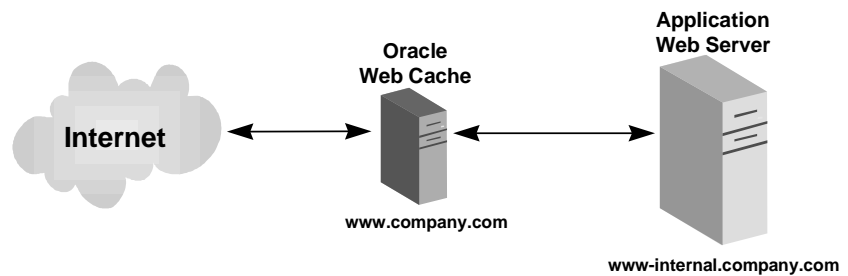


Figure 24: Oracle9iAS Web Cache on different node from Web server

A dedicated deployment of this nature is often preferable to the co-located deployment previously mentioned. In a dedicated scenario, there is no risk of resource contention with other server processes. Also note that Oracle Web Cache performs superbly on commodity hardware, so a dedicated deployment need not be a costly one in terms of hardware expenditure. For very high-volume Web sites, and to avoid a single point of failure, two or more nodes running Oracle Web Cache may be deployed behind a third-party network load balancing device.

Multiple Servers with Oracle9iAS Database Cache

This category of deployments adds Oracle9iAS Database Cache to the mix, allowing for even greater data and application scalability by offloading processing from the back end database. Figure 25 and Figure 26 illustrate just two deployment examples. Certainly, many alternative variations are possible.

Note again that the dotted lines in the figures represent just one suggestion of where a firewall may be placed within the configuration. An Oracle9iAS user will deploy a firewall according to various user-specific requirements and vulnerabilities. For example, a user who maintains sensitive data in an Oracle9iAS Database Cache instance will make sure that it is behind the firewall. Alternatively, a user that only caches public data may prefer to put the cache node in front of the firewall to enhance responsiveness.

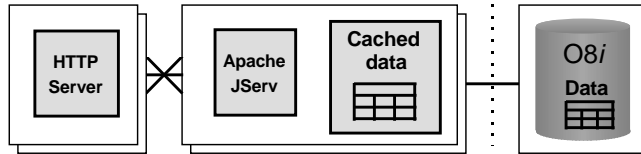


Figure 25: JSP/servlet application with separate Oracle HTTP Server tiers, Apache JServ and Oracle9iAS Database Cache tiers. This configuration introduces scalable data access to the system. The back end database will serve more users as data queries can be served directly from the middle tier caches.

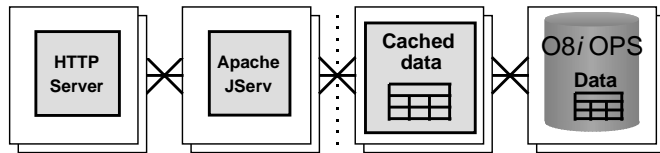
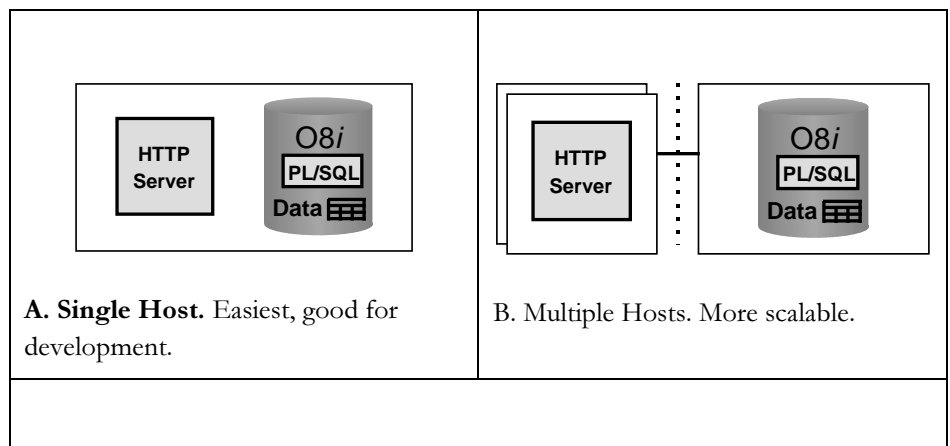


Figure 26: JSP/servlet application with separate Oracle HTTP Server tiers, Apache JServ tiers, Oracle9iAS Database Cache tiers, and Oracle Parallel Server (OPS). There is no single point of failure in this configuration, as even the database node is made redundant with OPS. The configuration provides high scalability, fault-tolerance, and availability.

PL/SQL Application

Deployment of PL/SQL applications offers similar options and benefits as those of the JSP/servlet applications discussed above. One notable difference, however, is that PL/SQL code will run as stored procedures within the database process. As illustrated in Figure 27, the addition of Oracle9iAS Database Cache allows this PL/SQL code to be replicated to the middle tier for execution. This can often boost application scalability dramatically and off load CPU cycles from the back end database node.



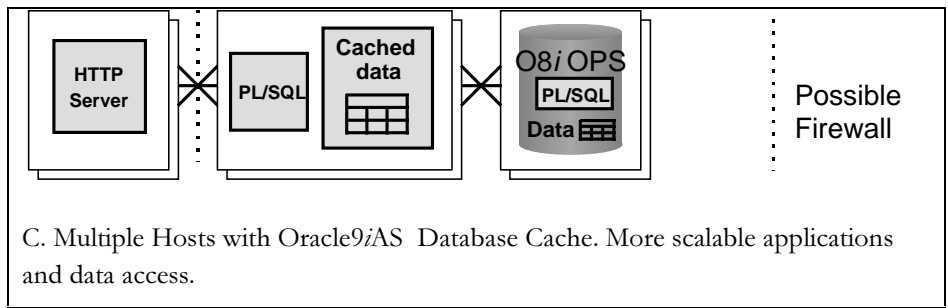


Figure 27: PL/SQL application deployment options.

SUMMARY

Oracle9iAS offers high levels of scalability, availability, and load balancing.

Oracle9iAS can be deployed in a multitude of configurations, enabling you to re-deploy your applications for additional performance or reliability without needing to alter your application code.



Oracle9i Application Server
February 2001

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com

Oracle Corporation provides the software
that powers the internet.

Oracle is a registered trademark of Oracle Corporation. Various
product and service names referenced herein may be trademarks
of Oracle Corporation. All other product and service names
mentioned may be trademarks of their respective owners.

Copyright © 2000 Oracle Corporation
All rights reserved.