Oracle9i Application Server: Availability, Scalability, and Manageability of J2EE and Web Cache Clusters

An Oracle White Paper
May 2002
# Oracle9i Application Server: Availability, Scalability, and Manageability of J2EE and Web Cache Clusters

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction to Oracle9i AS and Clustering</td>
<td>5</td>
</tr>
<tr>
<td>Clustering: An Introduction</td>
<td>5</td>
</tr>
<tr>
<td>Audience</td>
<td>5</td>
</tr>
<tr>
<td>Oracle9i AS (J2EE and Web Cache Install)</td>
<td>5</td>
</tr>
<tr>
<td>Document Organization</td>
<td>6</td>
</tr>
<tr>
<td>Clustering Characteristics</td>
<td>6</td>
</tr>
<tr>
<td>Availability</td>
<td>6</td>
</tr>
<tr>
<td>Scalability</td>
<td>7</td>
</tr>
<tr>
<td>Manageability</td>
<td>7</td>
</tr>
<tr>
<td>Clustering Overhead</td>
<td>8</td>
</tr>
<tr>
<td>Alternatives</td>
<td>9</td>
</tr>
<tr>
<td>Summary</td>
<td>9</td>
</tr>
<tr>
<td>Oracle9i AS Clustering Framework</td>
<td>10</td>
</tr>
<tr>
<td>The Terminology</td>
<td>10</td>
</tr>
<tr>
<td>The Clustering Framework</td>
<td>10</td>
</tr>
<tr>
<td>OPMN – Oracle Process Management and Notification Service</td>
<td>10</td>
</tr>
<tr>
<td>Process Management</td>
<td>11</td>
</tr>
<tr>
<td>Process Notification</td>
<td>11</td>
</tr>
<tr>
<td>Infrastructure Repository</td>
<td>11</td>
</tr>
<tr>
<td>DCM – Distributed Configuration Management</td>
<td>11</td>
</tr>
<tr>
<td>Configuration Versioning and Checkpointing</td>
<td>12</td>
</tr>
<tr>
<td>Example: Joining a New Node to a Cluster</td>
<td>12</td>
</tr>
<tr>
<td>mod_oc4j</td>
<td>12</td>
</tr>
<tr>
<td>Identifying the requests to Route</td>
<td>12</td>
</tr>
<tr>
<td>Identifying which OC4J to Route the request to</td>
<td>12</td>
</tr>
<tr>
<td>Communicating with OC4J</td>
<td>13</td>
</tr>
<tr>
<td>Transparent Updates to mod_oc4j Routing Table</td>
<td>13</td>
</tr>
<tr>
<td>Oracle9i AS Clustering Framework Summary</td>
<td>13</td>
</tr>
<tr>
<td>Oracle9i AS Web Cache Clustering</td>
<td>14</td>
</tr>
<tr>
<td>Overview of Cache Clusters</td>
<td>14</td>
</tr>
<tr>
<td>Clustering Characteristics for Web Cache Clusters</td>
<td>14</td>
</tr>
<tr>
<td>Availability: Transparent Failover</td>
<td>14</td>
</tr>
<tr>
<td>Availability: Quick Automated Restarts</td>
<td>15</td>
</tr>
<tr>
<td>Availability: Shared “Cluster” State</td>
<td>15</td>
</tr>
<tr>
<td>Availability: Graceful Degradation</td>
<td>16</td>
</tr>
<tr>
<td>Availability: Single Point of Failure</td>
<td>16</td>
</tr>
<tr>
<td>Scalability: Load Balancing and Distributed Caching</td>
<td>16</td>
</tr>
<tr>
<td>Manageability: Cluster Deployment</td>
<td>16</td>
</tr>
<tr>
<td>Manageability: Cluster Re-Configuration</td>
<td>16</td>
</tr>
<tr>
<td>Clustering Overhead: Security</td>
<td>16</td>
</tr>
</tbody>
</table>
Clustering Overhead: Communications ........................................ 17
Clustering Overhead: Performance ........................................... 17
Web Cache Clustering Summary .............................................. 17
Oracle9iAS J2EE (Web Applications - JSP, Servlets) Clustering .... 18
Islands ................................................................. 18
Sample Scenario - Clusters, Instances, and Islands ............. 18
Session State Replication .................................................. 19
The Replication Overhead .................................................. 19
Integration with mod_oc4j ................................................ 20
JSP and Servlet Clustering Evaluation ................................ 20
JSP and Servlet Clustering Summary ................................ 20
Oracle9iAS Java Object Cache .......................................... 21
Java Object Cache Summary .............................................. 21
Oracle9iAS J2EE (EJB) Clustering ................................... 22
EJB Clustering Overview .................................................. 22
Clustering Characteristics for EJB Clustering ....................... 22
  Availability: Transparent Failover .................................. 22
  Availability: Quick Automated Restarts ......................... 22
  Availability: Shared “Cluster” State ............................. 22
  Availability: Graceful Degradation ............................... 24
  Availability: Any Single Points of Failure .................... 24
Scalability: Load Balancing ............................................. 24
Scalability: Distributed Caching ....................................... 24
Manageability: Cluster Re-configuration .............................. 24
EJB Clustering Summary .................................................. 24
Oracle9iAS OHS (Oracle HTTP Server) Clustering ............... 26
OHS Overview ........................................................... 26
Clustering Characteristics for OHS .................................... 26
  Availability: Transparent Failover ................................. 26
  Availability: Quick Restarts ....................................... 27
  Availability: Shared “Cluster” State ............................ 27
  Availability: Graceful Degradation ................................ 27
  Availability: Single Point of Failure ............................ 27
Scalability: Load Balancing ............................................. 28
Scalability: Distributed Caching ....................................... 28
Manageability .............................................................. 28
Summary of OHS Clustering .............................................. 28
Oracle9iAS Clustered Deployment .................................... 29
3rd Party Component Certification ..................................... 29
Listeners ................................................................. 29
Firewalls and Load Balancers .......................................... 29
Security and Certificate Authority ..................................... 29
Clustered Deployment Scenarios ....................................... 30
Loosely Coupled Clustering ............................................. 30
Description ................................................................. 30
Clustering Characteristics of Loosely Coupled Cluster .......... 31
**INTRODUCTION TO ORACLE9/AS AND CLUSTERING**

**Clustering: An Introduction**

Clustering makes several independent systems appear like one. Together, the clustered system is more scalable than an individual system. The clustered system masks the failure of a member - and hence is more available.

This paper first establishes the salient characteristics to look for in any clustered system. It then discusses how several systems running 9iAS Release 2 can be clustered together for better scalability, availability, and manageability.

**Audience**

This paper is intended for architects and designers. It focuses on the features and implications of different clustering features and deployment scenarios.

The 9iAS product documentation covers the configuration steps, command syntax etc. required to implement the configurations discussed in this document.

**Oracle9iAS (J2EE and Web Cache Install): An Introduction**

This paper focuses on the ‘core’ components of Oracle9iAS Release2. Hence, a reference to Oracle9AS in this paper in general is a reference to Oracle9iAS Release 2 J2EE and Web Cache Install.

The components that fall in the core category are:

- **Web Cache:** This is typically the first component of Oracle9iAS to receive the request. For both static and dynamic requests, it can cache the result and then replay the results, thus reducing the workload of the machines behind. In addition, these Web Cache instances can themselves be clustered.

- **Oracle HTTP Server (OHS):** This is the next in line after Web Cache to receive a request - this sub-system comprises a web server (based on Apache), a perl execution environment, and a PLSQL and OC4J routing system.

- **Oracle Containers for J2EE (OC4J):** This is the J2EE compliant container in Oracle9iAS. It provides clustering capabilities for the J2EE components - Servlets, JSP, and EJB. It also contains other mechanisms, such as Java Object Cache, which provides distributed caching capabilities.

- **Oracle Enterprise Manager:** Oracle Enterprise Manager provides the web-based administration system of Oracle9iAS Release 2.
Infrastructure Repository: All the configuration and deployment information is stored in a repository. This is referred to as an infrastructure repository. It may be a database or may just be a file system.

Clustering Framework: There are some new components that work in tandem to provide significant new clustering capabilities in 9iAS Release 2. We cover these new additions in the next section.

Document Organization
This paper’s goal is to evaluate and understand the impact of Oracle9iAS Release 2 clustering. It is organized as follows:

• We first discuss some common criteria / characteristics of clustering.
• Then we get in depth to understand each component’s clustering capabilities and evaluate it against the criteria established.
• Next, we take the core components and put them together into some common deployment architectures.

Thus, the reader will be able to have an in-depth understanding of the pros and cons of the different clustering scenarios.

Clustering Characteristics
This section discusses the generally accepted (or expected!) characteristics of a clustered system. We will later evaluate each component’s clustering capability against these criteria.

Availability
Clustered system are supposed to have better availability than an individual member. Within availability, there are different facets:

*Availability: Transparent Failover*
Transparent failover enables a member of a clustered system to take the place of another member of the system, without the end-user knowing of such a change. This transparency provides an impression of a continuously available system, while allowing for individual member downtime – either planned or unplanned.

*Availability: Quick Automated Restarts*
Identifying and restarting a “down” member of the cluster improves availability. Without this an administrator intervention will be required, which increases the restart time and thus may reduce availability.

*Availability: Shared “Cluster” State*
For a cluster member to take the place of another member in the cluster, the relevant state of requests (or service) needs to be shared across all members. With
this sharing of state, the end-user does not perceive any break in his service as new cluster member take the task of servicing the same client.

**Availability: Graceful Degradation**

Graceful degradation refers to providing a reduced level of service (instead of a total failure) in exception situations. An example might be if a machine is down. There is a cost associated with this and we will discuss the associated cost-benefit tradeoffs.

**Availability: Single Point of Failure**

In general, single points of failure are bad – especially if that point has lower availability statistic than the rest of the system. Hence, it is important for a clustering solution to present deployment choices to avoid single points of failure.

**Scalability**

Clustering does not always improve performance of the system, but it should enable ‘scaling’ the setup to support more users. The different facets of scalability are:

**Scalability: Load Balancing**

Without a fair and good load-balancing algorithm, the clustered systems may starve or be overloaded, thus reducing the effectiveness of clustering.

**Scalability: Distributed Caching**

A distributed caching mechanism allows computation done by one member in the cluster be available to other members. Thus every member benefits and is able to provide a better performance than the individual system.

This is different from the shared cluster state – the content in a distributed cache can be re-created from the source at the cost of performance, while the content in the in-memory state may be lost if it is not replicated. Thus distributed cache impacts performance while shared cluster state impacts availability.

**Manageability**

Typically, adding more machines to the cluster increases the amount of effort required to manage them. Hence, it is important for clustering solutions to provide tools to manage the cluster.

**Manageability: Rolling Deployment**

A configuration change (or application deployment) made to one machine may need to be transparently and automatically propagated across all members. This can either be propagated immediately or in a rolling fashion.
Manageability: Changing Cluster Membership

The ease of adding or removing members from the cluster and their automatic re-configuration is important. A cluster-wide downtime to change cluster membership will clearly be unacceptable.

Other Criteria

We do not discuss some other manageability criteria in this paper. Specifically, we do not discuss rolling upgrade, which involves upgrading the clustering software itself. We also do not cover notification such as paging, email etc. which keeps the administrator abreast of the cluster state. Most 9iAS components use log files to log the state change events.

Clustering Overhead

The members of a cluster generally need to incur some additional communication and store more meta-data / information about each other. It is important to understand this overhead for cost-benefit purposes.

Clustering Overhead: Security Impact

Security generally gets harder as more machines are added to a cluster. This is simply because there are that many doors to close. Moreover, in addition to the applications, the cluster communications and setup also needs to be secured.

Clustering Overhead: Communications and Performance

The chattiness of the clustering infrastructure determines the communication overhead. Keeping multiple systems in sync also has a synchronization overhead that may manifest itself as performance degradation to the end user. Hence these overheads must be kept low.

Clustering Overhead: Cost

This typically refers to the dollar cost of the deployed system - including product licensing and other products required to make clustering work. Oracle9iAS Release 2 has no additional licensing cost for clustering - however, from a deployment perspective we will talk to the overall system when appropriate.

Clustering Overhead: Other Criteria

If clustering requires the systems to be "next" to each other, it may not provide fail-over in case of natural disasters. Also, in some cases a cluster distribution over WAN may result in better performance to the end user. This paper however does not discuss proximity of clustered systems as a criterion.

Another criterion that is omitted for now is the time lag in distributing information in the cluster. If this time lag is long then the risk to availability (transparent failover) is high. In most cases in Oracle9iAS, this time lag can be controlled through configuration files, or, is relatively low.
Alternatives

Typical alternatives to clustering are – (i) use a machine with larger horsepower (ex. more CPUs), or (ii) use a front end load balancer, or (iii) use a clustered operating system. We discuss these scenarios when appropriate to help the reader with the cost-benefit analysis.

Summary

This chapter introduced Oracle9iAS (core install) and defined the characteristics of a clustered system.

The next few chapters walk through the individual components in Oracle9iAS (core install) and discuss their clustering capabilities.
Oracle9iAS Clustering Framework

In Release 2, Oracle9iAS introduces some new terms and a new framework for clustering.

The Terminology

**Oracle9iAS Instance** is a collection of component processes, spawned from the same 'Oracle Home'.

A **Cluster** is a collection of Oracle9iAS instances, all configured identically.

A **Node** is a server machine. It could theoretically run multiple Oracle9iAS instances that may or may not be part of the same cluster.

A **Component Instance** is a set of identically configured component processes within an Oracle9iAS instance.

An **island** is a group of processes in the cluster, which replicate session state amongst each other. This, in conjunction with mod_oc4j, enables transparent failover for the client.

The Clustering Framework

The different components that constitute the clustering framework are:

**Oracle Process Manager and Notification Server** (OPMN) manages (starts/stops/monitors) the processes within a 9iAS instance and channels events between the instances.

**Distributed Configuration Management** (DCM) distributes the configuration information across the components in the cluster and also saves it to the repository.

**Infrastructure Repository** contains all the relevant configuration information. All instances that have their configuration in the same repository are said to be members of a farm. The repository can be a database or just be file based.

**Mod_oc4j** plugs into Oracle HTTP Server (OHS) and routes to all OC4J instances using AJP. Its routing table is continuously kept updated by OPMN. Thus OPMN provides fault tolerance, DCM and infrastructure repository provide distributed deployment, and mod_oc4j provides load balancing.

Now, we will cover each of these components in more detail.

**OPMN** – **Oracle Process Management and Notification Service**

Every 9iAS instance has one OPMN instance. OPMN provides process management and notification services for all processes in that 9iAS instance.

---

1 This is Apache JServ Protocol, version 1.3.
Process Management

OPMN can manage (start/stop/monitor) processes it knows about - including external ones that may not be related to 9iAS in any way.

OPMN uses a combination of ping, reverse ping, and system specific process id information to monitor a process. If it determines it is dead, it restarts the process. External processes that may not have been instrumented for a reverse ping (heart-beat), are also adequately monitored using the ping and pid mechanisms.

In addition, when an instrumented process (specifically, OHS and OC4J) is down, internal book-keeping and notification ensures a cluster wide update of the state. Thus, mod_oc4j always contains updated routing information.

There is also an OPMN buddy process that monitors and restarts OPMN if required. The buddy process itself is not monitored.

Process Notification

OPMN also provides a simple, lightweight publish/subscribe notification system. It is leveraged by other components (such as DCM) to communicate with each other across instances in the cluster. OPMN is the sole means of communication across instances.

All notification is point to point, using HTTP to all instances in the farm. On receiving a notification, OPMN filters out the ones not targeted for that cluster.

Infrastructure Repository

All of the configuration information of a core installation is stored in the repository. This repository can either be a database or be file based. A database provides more flexibility and power.

DCM – Distributed Configuration Management

DCM - Distributed Configuration Management - provides the capability to distribute configuration across the cluster, ensuring all instances in the cluster remain in sync:

- It first validates a configuration change (including XML files within an EAR), abstracts out the instance specific information and then updates the repository with the changes.
• DCM then executes the change i.e. generates the configuration files required for different components and then applies those changes. OPMN may restart components if required.

• DCM then notifies all other instances in the cluster of the change, so they can update their configuration from the repository.

• On the receipt of the change notification, the receiving DCM gets the configuration from the repository and applies the change locally.

**Configuration Versioning and Checkpointing**

DCM enables versions of configurations – thus you can have a production configuration and a beta configuration, both residing in the same repository. This allows for a quick change of configuration of any 9iAS instance to match any other instance (or cluster) in the farm. The configuration version is saved either at an instance level or at a cluster level.

A checkpointing capability also allows for updating the repository with the actual runtime configuration of the system. Thus, any manual updates made to the configuration files can be captured in the repository.

**Example: Joining a New Node to a Cluster**

This feature allows you to add a new Oracle9iAS instance (typically, on a new node), to an existing cluster.

A “join cluster” trigger (either via OEM or command line) is sent to DCM on the new instance, which picks the cluster version of the configuration from the repository, and applies that to the 9iAS instance. No administrative intervention is required.

**mod_oc4j**

mod_oc4j sits within Oracle HTTP Server and (i) identifies the requests it needs to act on, (ii) determines which OC4J to route those requests to, and (iii) communicates with that OC4J process.

**Identifying the requests to Route**

Every J2EE based (web) application, when deployed, needs to be associated with a root context. This root context (i.e. a URL prefix) acts as the identifier of requests that need to be handled by mod_oc4j.

**Identifying which OC4J to Route the request to**

An island is a set of OC4J processes within the cluster that replicate session information among each other. The J2EE section of the paper goes in more
depth on the island concept.

For the purposes of this routing discussion, it is important to understand that once a (stateful) request is routed to any member of an island, it cannot be later failed-over to an OC4J process outside the island.

1. The mod_oc4j routing algorithm for stateless requests is round robin. If this is a new request (with no valid routing cookie), it picks the next process from the list, and picks that as the destination.

2. If an attempt to send the request to that process fails, mod_oc4j picks the next process from the list, until all processes in that OC4J instance are exhausted. And if so, it returns failure to the client.

3. If it is a request with a valid routing cookie, the cookie already identifies the process to route to uniquely. This target process is changed only if the target OC4J process is dead. In that case, mod_oc4j picks the next process from the same island and routes to it.

Communicating with OC4J

Mod_oc4j now extracts some relevant parameters (ex. SSL info, certain environment variables etc.) and forwards them over the persistent connection to OC4J, using AJP13 protocol.

Mod_oc4j analyzes the response from OC4J and takes appropriate actions - ex. if a single sign on redirect is required.

Transparent Updates to mod_oc4j Routing Table

OPMN on all 9iAS instances in the farm notify each other of the up/down status of OC4J within their instance. In turn, every OPMN also notifies its local mod_oc4j of changes in the OC4J status on all machines within the cluster. This allows mod_oc4j to keep its routing table updated, without any intervention from administrator.

Oracle 9iAS Clustering Framework Summary

OPMN (Oracle Process Manager and Notification Service) provides fault tolerance, DCM (Distributed Configuration Manager) and infrastructure repository provide distributed deployment, and mod_oc4j provides load balancing.

The next few chapters leverage these components along with some additional concepts to build a highly available, scalable deployment architecture.
ORACLE9/AS WEB CACHE CLUSTERING

Web Cache component provides a way to cache both static and dynamic requests and thus provides performance boost by reducing the need to go to the back end server (cache hits). Since fewer requests go to the back-end application server, it also results in better scaling of the back-end systems.

Overview of Cache Clusters

In a cache cluster, multiple instances of Oracle9AS Web Cache operate as one logical cache. Each cluster member can propagate object changes - including invalidations, destroys, and replaces - through the cluster’s messaging system to other caches in the cluster. This allows for the content to stay synchronized, without the overhead of centralized control.

The above figure shows an Oracle9AS Web Cache cluster that contains three cache cluster members. As the figure shows, the cluster members communicate with one another as well as with the application Web servers and with the client. An external load balancer is required to route requests from the client to across the cluster.

The Web Cache clustering system incorporates all capabilities of OPMN and DCM discussed in the previous section – albeit with some implementation differences.

Clustering Characteristics for Web Cache Clusters

In this section, we evaluate the Web Cache clusters against the clustering characteristics of the first chapter. The goal is to further understand how Web Cache clusters work to provide certain clustering features.

Availability: Transparent Failover

To get data from any Web Cache, you need to know the host name (or IP address) and the port number to access. If Web Cache becomes unavailable on that host/port, or, if that host becomes unreachable, the failure is visible.

A clustered Web Cache can mask the failure of a member by either fetching the contents from another member, or, by going to the back-end server. However, one still needs to prevent the point of entry to be a single point of failure. This
can be achieved in two ways:

**Leverage External Load Balancer:** These hardware load balancers regularly ping the backend servers (in this case, all members of the Web Cache cluster). If they see a failure, they will not route to the Web Cache member that is down. Most load balancers themselves have special hardware level mechanisms to cluster them while providing the same IP or hostname to the end-user.

**Leverage OS Clustering:** In this scenario the operating system takes care of having two machines (and thus two distinct Web Cache processes) appear to the end user at a single IP address and port. Thus the OS makes sure if one of the machines fails, it will transparently transfer the request to the second machine, and thus the failure is not exposed to the end user.

While the above approaches can be done even without clustering Web Cache, there are advantages to clustering the cache, which will be evident as we discuss more clustering characteristics. The section on shared cluster state has more discussion on transparent fail-over.

**Availability: Quick Automated Restarts**

Web Cache contains an auto restart process, which pings the Web Cache process at regular intervals to ensure it is up and running. In case it is not, the auto restart process restarts Web Cache. This facility is available independent of clustering multiple instances of Web Cache. There is no monitoring for the restart process itself.

**Availability: Shared “Cluster” State**

A capacity driven algorithm enables partitioning of content ownership to happen without any synchronization between the cluster members. On a cache miss, a cluster member fetches content either from the origin servers (referred to as the owned content) or from other cluster members (referred to as the on-demand content).

This partitioning provides a better cluster cache hit ratio (i.e. the content is fetched from the cluster) despite local cache misses (i.e. the cache did not have it in its memory). Once the cache miss is satisfied (whether through the origin server, or through another member of the cluster), the content is cached to provide a better local cache hit ratio.

Thus the cluster state in a Web Cache cluster is – (i) the partitioning scheme, (ii) the configuration (which tells it about cluster members), and (ii) the cached content. Since either is easily reproduced or available to all members of the cluster, in the case of a member failure, the end-users do not see a performance or availability\(^2\) impact.

---

\(^2\) This assumes the front-end load balancer takes out the Web Cache member that is
Availability: Graceful Degradation

As discussed in the shared cluster state section, the partitioned cache results in better cluster-hit ratio despite local cache misses. This improves (i.e. reduces) the performance degradation that might otherwise be observed due to the failure of a cluster member.

It should be noted that a Web Cache cluster member (re) starts cold, and has to re-build its cache content through the first few requests. Also, on failure, there are fewer instances of Web Cache running, and this will impact performance until the instance comes back up.

Availability: Single Point of Failure

The back end server can reproduce any content in Web Cache or in the Web Cache cluster. Thus, Web Cache is not a single point of failure in the system.

Scalability: Load Balancing and Distributed Caching

Cache misses of on-demand content result in a direct request to the cluster member that owns that content. However, in the case of a miss on an owned content, the Web Cache does capacity based load balancing across a set of back-end application servers. Thus, Web Cache makes sure that a server defined to have a higher capacity to service requests gets proportionately more requests than a smaller capacity server.

Manageability: Clustered Deployment

All members of the Web Cache cluster see the same rules and configuration. These rules can be changed on any cache and can be propagated across the cluster - thus providing a more manageable cluster wide deployment of changes. Moreover, this propagation can be immediate, or, when appropriate can be done in a rolling fashion too.

Rolling deployment causes a member to be removed from the old configuration cluster and moves it into a new configuration cluster - until all members are part of the new configuration cluster.

Manageability: Cluster Re-Configuration

A Web Cache cluster re-configuration - moving new members in and out of a cluster - does not cause a cluster wide downtime. It however does require the specific member impacted by the change to be restarted, so it can pick up the configuration of the new cluster.

Clustering Overhead: Security

The Web Cache cluster will typically have all its members within the same sub-net. By controlling the routers to prevent messages from outside the subnet on the down from its routing. Otherwise, despite shared cluster state, the impact will be visible.
cluster notification ports, one can easily prevent a malicious machine from spoofing as a cluster member. If required (but not recommended for performance reasons), https can be used for peer notification.

Since the Web Cache cluster will typically be deployed in a DMZ, with no internal traffic visible outside the DMZ, snooping of data over the wire to determine cache contents is less of a concern.

**Clustering Overhead: Communications**

The Web Cache members are event driven in their communication with each other. There is no heartbeat mechanism among them, and thus there is not much communication overhead among them in a steady state.

The events that cause communication are typically a cache miss, or cache invalidation or rules re-configuration message that needs to be propagated to all members, or if a member notices that another Web Cache is down it notifies all members of the cluster of this state.

Invalidation events sent by the backend server to invalidate a cached content can be configured to go to a single member, or, to all members of the cluster separately. In the former case, the member propagates the event information to all members in the cluster.

**Clustering Overhead: Performance**

- Since the records indicating who owns the different contents are available locally to every member, the lookup of who to go for data is quick.
- Next, each individual Web Cache is allowed to cache all data – depending on its capacity, thus the cache misses in general are reduced.
- A Web Cache keeps its connection to the origin servers and other cluster members open for longer intervals using keep-alive. This amortizes the performance overhead associated with establishing a communication channel.

Thus, the performance overhead caused by clustering the Web Cache systems is quite low. On the other hand, the performance boost provided by such clustering can be significant.

**Web Cache Clustering Summary**

Web Cache provides significant performance improvements even without any clustering due to better cluster cache hit ratio even on local cache misses. Clustering also hides individual member failures, thus improving availability. The single click configuration updates across the cluster also provide improved manageability.
ORACLE9/AS J2EE (WEB APPLICATIONS - JSP, SERVLETS) CLUSTERING

This section focuses on J2EE web applications; EJB clustering is discussed separately.

Islands

As discussed earlier, an OC4J instance is a group of identically configured processes within a 9iAS instance. An island is a group of OC4J processes in a cluster of OC4J instances that replicate session state amongst each other and thus back each other up in the event of a failure. Thus, these processes have to have the same configuration, which is ensured by making these processes part of the same OC4J instance.

Adding a lot of processes in an island is not very advantageous, since after about 2-3 processes the availability does not improve by much, and the overhead of keeping the state of these processes in sync grows linearly.

Sample Scenario – Clusters, Instances, and Islands

The adjacent figure demonstrates the concepts further.

This picture has four 9iAS instances, each with two OC4J instances – Beta and Production.

The Beta OC4J instance has one island Is1, containing process #1 and process #6, on only two of the four 9iAS instances. mod_oc4j will not route any requests to the 9iAS instances without a process in the Beta instance. There is just one island in the Beta instance – Is1 - and the state will be replicated across them.

The Prod instance cluster has processes associated with it on all 9iAS instances, and contains three islands – Is2 through Is4. Is2 is well configured since it has a

3 These instance names are an example only. This paper does not cover the pros and cons of actually deploying beta and production software on the same nodes / 9iAS instances.
process on another node that it can failover to. Is3 provides even better failover – since its constituent processes span three nodes. However, the replication cost is higher for Is3, since the replication has to be done across three processes. Is4 however does not provide a node level failover – it provides only a process level failover (between processes 7 and 8).

The load balancers and Web Cache could be configured without requiring session stickiness – since mod_oc4j handles the session related routing. Thus, the external load balancers can route even stateful requests to any of the backend 9iAS instances. Also, it is recommended that Web Cache be clustered. However, clustering of Web Cache has no impact on the rest of the system.

**Session State Replication**

Different HTTP requests from the same client may be ‘bound’ together with a cookie. This cookie contains sufficient information to provide continuity between requests.

If the server dies the state associated with a request (i.e. cookie) may be lost. There are three choices to guard against this:

- **State Safe Applications** save the state in the database (or other persistent store), thereby avoiding the loss of state on server crash. However, there is a performance penalty to do these writes.

- **Stateless Applications** do not have state that needs to be carried between requests, and hence are not impacted by a server crash. Any other server is able to handle the request.

- **Stateful Applications** benefit from the OC4J session state replication feature, which replicates the session state across all islands in the cluster automatically. This is done for both JSP and Servlets.

OC4J replicates the state automatically (using multicast) when the session is updated (i.e. setAttribute() is called). The replication is best effort – if it is important to guarantee it the state safe development model should be used.

**The Replication Overhead**

**Network Overhead** - Since the replication of state is done using multicast, the overhead is independent of the number of processes in the island. It is of course dependent on how often the state gets changed, since each state change causes a message on the wire.

Hence, it is recommended to put coarse objects and not small member objects in a session. Example: Including a Person object is much better than having firstName, lastName, address, phone etc. since for each update of these members the session multicast will happen.

**Memory Overhead** - Since all processes in the island have to have the sessions from
all processes in the island, the memory associated with this session storage in each
process grows linearly as the number of processes in the island grows. Hence, it is
a good practice to: a) keep the session state small, and b) keep the number of
processes in the island small (to 2 or 3). This provides sufficient availability while
reducing the drain on performance due to large and frequent replications.

Specifically, if a session state is 1/2MB, and there are 4 processes in an island,
and each process has 250 live sessions at any point, then each process will
consume: 1/2MB * 4 * 250 = 500MB for the sessions. However, reducing this
into two distinct islands reduces the memory consumption: 1/2MB *2 * 250 =
250MB (In a steady state, the incoming sessions are divided evenly across the
two islands).

This also has an impact on the CPU usage etc., since the memory replication does
take CPU cycles to take the data off the wire and put it in the replicated state
table.

Thus, the instances, islands, and processes can be tuned correctly for optimum
performance.

**Integration with mod_oc4j**

mod_oc4j understands the concept of island and that all processes in an island, not
instance or cluster, are candidates for fail-over. Thus, when it does need to re-
route a request to a backup node, it picks an alternate process from the same
island (transparent to the end user) and routes the request.

**JSP and Servlet Clustering Evaluation**

The impact of JSP / clustering, along with some interesting deployment scenarios,
is discussed in detail in the deployment section of this paper. Hence, that
information is not repeated in this section.

**JSP and Servlet Clustering Summary**

With the concept of instances and islands, scalability and availability of J2EE web
applications can be tuned better. The session state replication overhead can also
better controlled. Also, mod_oc4j re-routes the request to another member of the
island on failure, thus providing improved availability.
Oracle9iAS Java Object Cache

Java Object Cache in OC4J enables data replication amongst processes even if they have no island, instance, or cluster relationship amongst each other. This replication provides performance improvement by caching expensive (shared) java objects – no matter which application produced them, and also provides availability improvement in case the sources (ex. Database, an external application etc.) required to re-create the Java objects are down. Moreover, the Java Object Cache also supports versioning of objects, thus allowing different applications to have different versions of an object, which is especially useful during application upgrades.

The Java Object Cache is an “in-process” cache of Java objects that can be used on any Java platform and by any Java application. The Java Object Cache allows applications to share objects across requests, across users and coordinates the life cycle of the objects across processes.

It supports two modes of operation, local mode and distributed mode. Using local mode, objects are isolated to a single Java VM process and are not shared. Using distributed mode, the Java Object Cache can propagate object changes - including invalidations, destroys, and replaces- through the cache's messaging system to other communicating caches running either on a single system or across a network (the Java Object Cache messaging system is built on top of TCP/IP).

If an object is updated or invalidated in one process, it is also updated or invalidated in all other associated processes. This distributed management allows a system of processes to stay synchronized, without the overhead of centralized control.

Java Object Cache Summary

The goal of the JSP / Servlet and even EJB clusters (to be discussed next) is to provide a more load balanced and fault tolerant system. On the other hand, Java Object Cache focuses on providing better performance by distributing pre-computed objects across the servers so that an expensive computation is done only once.

This caching does indeed have the side effect of improved scalability and availability, since the contents of the cache are available for results computation even when the back-end server is down.
ORACLE9/AS J2EE (EJB) CLUSTERING

EJB Clustering Overview
An EJB cluster is a subset of an OC4J island, with some added security to limit replication only to that island. The EJB client stubs provide load balancing and fail-over across the EJB cluster. Thus, the stubs perform the role that mod_oc4j performed in the web applications context.

The client application can choose to leverage only a few processes of the island for load balancing by explicitly specifying them, or, may choose to gain better fault tolerance by looking up this list dynamically.

Static Retrieval - The lookup URL specifies the JNDI addresses of all OC4J nodes that should be contacted for load balancing and fail-over.

Dynamic Retrieval - The client (proxy) performs a special lookup with a "lookup:" prefix. All OC4J processes that can be contacted for load balancing and fail-over are dynamically discovered during this call.

Clustering Characteristics for EJB Clustering
Now we will evaluate EJB clustering against the standard clustering characteristics we had established earlier.

Availability: Transparent Failover
The EJB client side proxy automatically tries a new server on failure. The list of servers to try is either supplied statically or is discovered dynamically during the first lookup call. In either case, the failover of the EJB method calls is transparent.

During dynamic discovery phase only a single is specified during lookup time. Hence, a failure of the lookup call (because the single referenced server is down), will result in the failure being visible by the EJB client.

More discussion on transparent fail-over is also provided under the shared cluster state section.

Availability: Quick Automated Restarts:
Since EJBs run in an OC4J server that is managed by 9iAS clustering framework, all the discussion in the OPMN section of clustering framework for OC4J process restarts, applies to EJB.

Specifically, if the OC4J process goes down, OPMN quickly re-starts it.

Availability: Shared “Cluster” State
The state of all beans is replicated at the end of every method call to all OC4J instances in the EJB cluster. This state of the bean is replicated to all nodes in the
cluster, using a JMS multicast topic. This state stays in the topic until it is needed. Then when a method call comes into an alternate OC4J instance, the latest state for the bean is found in the JMS topic, reinstated, and the bean invocation continues.

**Stateless Session Bean Clustering**

Stateless session beans do not require any state to be replicated among nodes in a cluster, but clustering them gives better load balancing and fault tolerance.

**Stateful Session Bean Clustering**

Stateful session beans require state to be replicated among nodes. This state may be replicated in one of two ways:

- *Replication at JVM termination:* The state of the stateful session bean is replicated to another server within the EJB cluster.\(^4\)

- *Stateful session context replication:* This is similar to the session state clustering for web applications discussed earlier. It is replicated at the end of each method call.

**Entity Bean Clustering:**

The state of the entity bean is saved in a persistent storage, such as a database. Thus, when the client loses the connection to one node in the cluster, it can switch to another node in the cluster without worrying about replication of the entity bean state.

However, other instances may have cached the state. Hence, the entity bean that changes state notifies other instances that their state is no longer in synch. Thus, everyone’s state is marked as *dirty*. The next time a request for that bean is load balanced to an instance with *dirty* state, the state is resynchronized to the "READ_COMMITTED" state within the database.

**Combination of HTTP and EJB Clustering**

If reference to an EJB is part of a web application (JSP / Servlet) state, it is important to cluster them both. Otherwise, even if the web application state is replicated, the *EJBReference* cannot be. It is generally a good practice to have the EJB cluster identical to the island for HTTP session replication. Thus, both states are replicated across the same set of processes, and the co-location provides performance improvement too.

---

\(^4\) If the proxy is using statically configured addresses, and does not specify all the servers in the EJB cluster, it is possible that the state replication may happen to a node not included in the statically configured addresses in the proxy. This could result in a perceived failure, but since the situation can be corrected with the right configuration, it is not covered here.
Availability: Graceful Degradation:

On failure of an EJB server, the proxy stub attempts to connect to the next available server in the cluster. This process takes time and thus the client may see degradation in performance as the proxy loops through the choices to find an available server.

However, this is a small price to pay to mask the failures from the EJB client.

Availability: Any Single Points of Failure

There are two possible situations when a single point of failure may manifest itself in the environment:

- If using JVM replication mode for the state, this may cause a single point of failure if the node itself goes down (e.g., Power failure).
- In the dynamic discovery mechanism, only a single server is provided for lookup – this may cause a failure to be visible if that server is down during the lookup phase.

Scalability: Load Balancing

The EJB proxy stubs running on the client side effectively load balance between the available backend EJB servers using a random selection algorithm, at each remote invocation request (and lookup too).

Co-location provides further performance improvements, and a co-located bean is preferred over one that will cause a network hop.

Scalability: Distributed Caching

The discussion on Java Object Cache in the J2EE Web Applications section applies here too. It can be used as a distributed cache, sharing expensive Java objects across the EJB cluster.

There is no other mechanism for distributed caching. Note that this is different from the distributed cluster state discussed in the Availability section.

Manageability: Cluster Re-configuration

Since the EJB clusters are essentially OC4J islands, the discussion on islands will apply here too. However, there is one difference that needs to be spelled out:

The client container never tries to rediscover the server addresses. Thus, dynamic re-configuration of an EJB cluster is not possible without re-issuing the lookup request from the EJB client. (If you recall, in the case of servlet requests, this was done automatically with notifications between mod_oc4j and OPMN).

EJB Clustering Summary

EJB clusters are essentially OC4J islands. The EJB proxy can be statically configured with the cluster members, or, it can look those up dynamically. The
proxy then randomly load-balances and fails over if required within the cluster members. In the case of stateful beans, the state is replicated across the cluster members either on JVM termination, or, at the end of each method call.

The individual members of the cluster continue to benefit through the quick restarts and monitoring capabilities, as well as the single click deployment capabilities of overall 9iAS.
ORACLE9/AS OHS (ORACLE HTTP SERVER) CLUSTERING

This chapter covers the non-J2EE aspects of OHS; a separate chapter is dedicated to J2EE clustering.

OHS provides an HTTP Server (powered by Apache) and dynamic content creation facility through mod_plsql, mod_perl, and other components. This section focuses on the impact of clustering on both these facilities.

OHS Overview

From an architecture perspective, OHS has a modular process oriented design:

• On Solaris, the web server has several processes – each one handling exactly one request at any given time. They share information (such as SSL keys, process state etc.) via a shared memory location.

• All the modules run within these processes. In some cases, such as mod_plsql, mod_oc4j, and mod_fastcgi, the modules are merely routers, sending the request to the correct back-end for processing.

Of these, Mod_plsql, mod_perl do not communicate across the processes even within the same OHS instance.

Mod_oc4j does share information across all OHS instances is and is not discussed in this chapter.

Clustering Characteristics for OHS

Availability: Transparent Failover

The failure of the web server is handled by an external component – a Web Cache or an external load balancer. These devices typically ping a back-end OHS and make sure that the server is up. If they find it to be down, they take it out of their routing table, and then add it back when it returns to service.

However, if a browser accessed OHS directly (DNS or IP address), the failure of OHS will be visible to the browser.

Assuming the external load balancer re-routes the request to a different OHS:

• In the case of static files (and other stateless requests), the other OHS simply services the request with no impact.

• In the case of a mod_plsql application, the other OHS invokes the mod_plsql component that handles the request. Since mod_plsql is state-safe i.e. the state is saved in the database the failover will remain transparent to the end-user.
• In the case of perl applications, there is a possible impact if the perl application retained state. The death of the OHS process will thus cause it to lose state.

• The impact on java applications is covered in a separate chapter.

• Transparent failover of C / C++ applications - If a pre-existing FastCGI process is not available, a new one is spawned.

• Transparent failover of SSL key exchanges is not an issue in case you have multiple OHS instances because SSL ends at the front-end load balancer. Thus, end-users do not have to incur the cost of re-negotiation of the key.

Availability: Quick Restarts
The clustering framework (specifically, the OPMN component discussed earlier) constantly pings the OHS process (es) and ensures they are up. If it detects a failure, it will ensure that all children are cleanly shutdown and then restart the parent OHS process.

Availability: Shared "Cluster" State
Within OHS there really is no "cluster" - each OHS is 'self-contained.' The external load balancer routes requests between these independent instances. There is no information shared between these unrelated OHS instances.

Mod_oc4j is an exception to this rule and has already been discussed separately.

The sharing of information or lack thereof and its impact is discussed adequately in the transparent failover section.

Availability: Graceful Degradation
OHS, the web server has almost no performance impact due to the death of a process, since it automatically restarts those processes very quickly. Such death may even be "planned" to recover any slow memory leaks in the process etc.

In the case of mod_plsql, the database connection is cached and reused across requests. When a process dies, this connection is lost. Thus, the connection(s) have to be re-established, slowing down future requests, until a steady state is reached. In case the entire OHS site (including parent process) is dead, the steady state may take a little while to attain.

Availability: Single Point of Failure
There is no identifiable single point of failure within the OHS component - it differs and depends on the type of request.

To the extent described in the "transparent failover" section, a failure of OHS process can result in lost state for some kinds of applications. This loss may
manifest itself as no impact to any end-user, or, may manifest itself as a failure perceived by several users bound to that process.

**Scalability: Load Balancing**

To load balance across OHS instances, an external load balancer such as Web Cache or a hardware load balancer is required. Within an instance, the following load balancing apply for the different programming approaches:

- **Java:** See the discussion on mod_oc4j in the J2EE section of this paper.

- **Static HTML:** Typically, the requests are served in a “round-robin” fashion by the OHS child processes. On NT, this round robin mechanism applies to the threads (since there is just a single child process). The parent does not interfere in the routing.

- **Perl:** Since mod_perl runs the Perl processes within the OHS child process, the load balancing that applies to static html also applies here.

- **PLSQL:** In addition to the static html load balancing described above, PLSQL can also leverage load balancing across the back-end databases. While Perl and other applications (ex. Java) that use OCI can also benefit from the TNS based load balancing available from a database cluster, in the case of mod_plsql (and PSP) based applications it also applies.

**Scalability: Distributed Caching**

OHS does not have facility for distributed caching of end user data. As discussed separately, that capability is part of the Java Object Cache.

**Manageability**

The benefits introduced by 9iAS clustering framework are applicable to OHS and hence not discussed separately.

**Summary of OHS Clustering**

Oracle HTTP Server generally requires a state safe or state less application for better availability. There is no provision for state replication as is the case with J2EE applications. However, the 9iAS clustering framework does provide fast restarts.

The loosely coupled deployment clustering scenario (see the deployment section for details) needs to be used when deploying OHS applications (Perl, PLSQL), since DCM does not support direct deployment of these. However, all the OHS related configuration is indeed replicated across the cluster, making manageability easier than before.
ORACLE9/AS CLUSTERED DEPLOYMENT

3rd Party Component Certification

Listeners
Oracle9iAS supports external web servers such as Microsoft’s IIS and Netscape’s iPlanet Web Server. A small component needs to be installed in these servers, and the backend clustering remains unchanged for the rest of Oracle9iAS.

It is feasible to configure (for example) iPlanet Web Server as a custom process managed by OPMN. In this case, OPMN will ensure this process is started when the Oracle9iAS instance starts, and it will monitor the process in a regular manner to ensure it stays up.

However, a regular heart beat mechanism is not available - so if the external listener process is up and running but is not responding to requests, OPMN will not restart it.

Firewalls and Load Balancers
Oracle9iAS is certified against several external load balancers and firewalls. The entire clustering infrastructure, routing algorithms have been tested with various combinations of these external devices.

The external devices are not managed by the clustering infrastructure discussed in this paper, but, these devices may be leveraged to provide additional capabilities to the clustered system. We will discuss these in the next section.

Security and Certificate Authority
The certificate authority usage for SSL server certificates, as expected, has no impact on the clustering.

Oracle9iAS includes a single sign on solution and the deployment scenarios discussed in the next section can all leverage this solution. The impact for the most part is some additional configuration and setup.
CLUSTERED DEPLOYMENT SCENARIOS

This section discusses the following deployment scenarios:

1. Loosely Coupled Cluster Configuration contains several stand-alone instances front-end’d by a load balancer (or Web Cache).

2. A 9iAS Cluster, which leverages all the new clustering capabilities of Release 2, and

3. Loosely coupled cluster with manual islands and routing – which is a mix between the previous two.

4. Multi-Cluster Deployment across Firewalls – provides a way to deploy 9iAS clusters across firewalls, which is useful if you want all OHS in DMZ and all OC4J behind a firewall.

More combinations are quite possible, but the intent is to provide sufficient discussion around the above three, so the reader may be able to leverage pieces from these scenarios to extrapolate to their own situation.

LOOSELY COUPLED CLUSTERING

This is the most common deployment model in the previous releases of Oracle9iAS, so we will talk about it first. Also, the 9iAS Release 2 clustering infrastructure is applicable only when all installations are of type 9iAS core, hence this is the only choice of clustering available for other installation types of 9iAS.

Description

In this scenario, a 3rd party load balancer or 9iAS Web Cache instance, routes to independent 9iAS instances at the back-end. These instances have no knowledge of each other. The Web Cache may be clustered, or, may be stand-alone.

The different instances may have independently configured number of OC4J processes. For this section, we will assume all configurations – including the number of processes – are identical across machines. From a 9iAS cluster infrastructure perspective, these are three independent 9iAS instances, and there is no 9iAS cluster defined that makes all three nodes above look as a single cluster.

The load balancer (or Web Cache) in the front is configured to load balance
requests across all nodes. Hence, all nodes have to have identical applications deployed on them. These nodes may or may not be running the same operating system.

Each of these instances (one on each node in this example) can be configured independently through OEM.

**Clustering Characteristics of Loosely Coupled Cluster**

**Availability: Transparent Failover**

Requests With No Existing Sessions: This case is the easiest – since there is no pre-existing session associated with the request, it can get routed to any of the backend nodes.

To ensure that a node that is down is not included in routing, almost all load balancers (including Web Cache), include a mechanism to ping a URL on each node to ensure they are up. If this URL does not return the desired data, that node is taken out of routing algorithm, and thus, the new http requests are never routed to a failed node. Thus, the end users never see the failure of a down system.

If the back-end node assigns a session id to the requests, the front-end load balancers will update the internal table to bind the session to that node. This is used in the next case when the request is re-sent.

Requests With Pre-existing Sessions: The load balancer, based on the cookie value, determines where to route the request. If the back-end node is up, it routes to that. Else, the load balancer will route to another node (depending on configuration settings). The following behavior is observed:

- If either member of the Web Cache cluster can satisfy the request, the failure of the back-end node is masked.
- In the case of a PLSQL application, the state is stored in database, so all they see is a possible performance hit (due to reconnection to the database from the other OHS processes). In the case of Perl or other types, it depends on the specific application – unless these were state safe or stateless applications, the end-user will see the failure.
- In the case of a J2EE application a failure will be visible – it can be avoided with some adjustment to this deployment scenario, as discussed in Loosely coupled cluster with manual islands and routing.

Failure During a Request: Many load balancers or 9iAS Web Cache will re-route requests if they cannot connect to the back-end server at all. However, if they are able to connect they do not re-route the request – and thus the end-user will see the failure.

Failure of Web Cache or Back-end Server: When Web Cache is deployed in a cluster
mode, it provides better cache hits and hence is able to better hide the failure of a back-end server. As discussed earlier, it is also able to hide the failure of individual members of the cluster by re-routing requests to other members or to the back-end server.

**Availability: Quick Automated Restarts**

The independent instances of Oracle9iAS all have the standard 9iAS clustering infrastructure components running on them.

OPMN continues to provide the quick un-assisted automated restarts of a down process within the instance. The change in state however is not communicated to other instances.

**Availability: Shared “Cluster” State**

As discussed earlier, J2EE web applications (JSP, Servlets) have their session state replicated across islands in the cluster. In this particular setup, we do not have a 9iAS cluster, but the replication will happen across the processes in the island within the same 9iAS instance. Hence, all instances should be configured to have at least two OC4J processes in an island on every 9iAS instance.

This still leaves open the case when the node itself goes down – since the state is not shared with other nodes. This situation is remedied with a slight change to the configuration, and is discussed in the deployment scenario: Loosely coupled cluster with manual islands and routing.

The cluster state is not shared for applications of other types – PLSQL, Perl etc. Thus, those application writers have to ensure state-safety (by persisting the state).

The Web Cache clusters will continue to function as a clustered cache independent of the fact that the back-end nodes are not clustered.

**Availability: Graceful Degradation**

If a node or OHS is down, the load balancer takes it out of circulation. This may cause a slow-down. However, this graceful degradation is only temporary, since the new clustering infrastructure in 9iAS will automatically restart the down instances.

In this particular deployment scenario, it must be noted that if only OC4J process is down, and not OHS or the node, the external load balancer will not know to re-route. Hence, the failure might be visible to the end-user. To avoid this situation, it is recommended to have at least two OC4J processes within an island, in each 9iAS instance. While this increases the replication overhead, it does provide a better level of fault tolerance.

**Availability: Any Single Points of Failure**

The independently installed and configured 9iAS instances are all replicas of each other and are able to fill in for each other in cases of failure.
Thus, from a system perspective, the front-end load balancer or the back-end database are the only potential candidates for single point of failure. The commercial load balancers solve this problem by actually including two machines that act like one - which ping each other for heartbeat. If one of the machines goes down, the other takes over.

For smaller to medium sites you can use a single instance of Web Cache as load balancer, but for larger sites a hardware load balancer is recommended. Auto restart mechanism of Web Cache keeps the downtimes small. Thus for small to medium sites, Web Cache and Web Cache clusters can be an effective substitute for higher cost load balancers.

Web Cache masks the failure of back-end servers, and a Web Cache cluster can mask the failure of a cluster member. However, when used as a standalone load balancer it is a possible single point of failure. This is because the browser sends its request to a specific IP address, and if the process (whether Web Cache or other process) on that host/port is down, the failure is visible.

An approach to avoid this failure, even without using the commercial load balancer packages, is to leverage clustering at the operating system level. In this case, the operating system makes two systems look like one - efficiently load balancing between them. Most server operating systems - Linux, Solaris, Windows 2000, all have support for clustering at this level.

**Scalability: Load Balancing**

There remain three layers of load balancing across the systems:

1. **Load Balancer (External or Web Cache)** - This is the most important load balancer for this kind of system. In most cases this algorithm is simple round robin, although, in the case of Web Cache, a capacity driven load-balancing algorithm is used to send requests to the back-end servers.

   When deployed in clustered mode, Web Cache also attempts to retrieve the data from the distributed cache across the Web Cache instances.

2. **OHS** - Each OHS instance has a pre-configured number of processes (UNIX) or threads (NT), which load balance via a shared semaphore mechanism. Thus, a long lasting request does not slow down other incoming requests. Since mod_plsql, mod_perl both run within the OHS process (or thread), the load balancing amongst them is performed at the OHS level.

3. **J2EE - OHS with mod_oc4j** - mod_oc4j performs the load balancing and routing across all back-end OC4J processes. However, in this particular deployment type, mod_oc4j knows only about the processes on that instance - thus, it does not load balance across all processes in the island.

---

5 Databases are not covered in this paper
but just those running on the same 9iAS instance.

**Scalability: Distributed Caching**

Oracle9iAS Release 2 has two distributed caching mechanisms – Web Cache cluster and Java Object Cache. Both are independent of 9iAS Clusters. Hence, these distributed caching mechanisms are available in this deployment scenario and can be leveraged for improved scalability.

**Manageability: Clustered and Rolling Deployment**

Since all the instances in this scenario are stand-alone and not clustered in the 9iAS sense, it is not possible to deploy applications to all the instances in one go. This increases the management overhead.

However, rolling deployment of these application deployments is certainly possible:

1. Those requiring the back-end machine restart (or OC4J / OHS restart) can be performed by first moving a machine out of the loosely coupled cluster (by re-configuring the front-end load balancer), performing the deployment, and then re-introducing it back in.

2. Those not requiring any restart (for ex. OC4J hot deployment) can just be deployed without moving out or into a cluster.

In both situations, the end-users should not be informed of the new URL (i.e. new application), until the upgrade of all machines is complete. In situations where an existing application is being upgraded with no change to the URL, a prudent strategy is to:

1. Break the existing cluster into two - and upgrade the new cluster with the new application
2. Test the new cluster
3. Re-configure the front-end machine to send requests to the new cluster of machines, and
4. Modify the old cluster machines and have them join the new cluster.

The terms remove from cluster and join into a cluster are discussed in the next section.

**Manageability: Cluster Re-Configuration**

In this deployment scenario, joining a cluster essentially implies the following actions:

1. Configure and test the new back-end system to be identical to the rest of the nodes in the system.
2. Add the new node to the front-end load balancer configuration file.

Removal from a cluster however only requires the front-end load balancer
configuration to be updated to not include the specific node in routing.

**Clustering Overhead: Security**
The security overhead of organizing a system this way is not much. Of course the number of systems added to the mix does linearly increase the expense of patching them (if required).

Typically the entire loosely coupled cluster will be behind the firewall and the Web Cache cluster or the load balancer will be in DMZ. The firewall needs to be configured correctly to allow requests only to the corresponding HTTP ports of these machines and only from the load balancer machine.

**Clustering Overhead: Performance and Communications**
The session replication feature uses multi-cast and thus the communications overhead is independent of the number of instances in the loosely coupled cluster. The replication does require extra memory to be reserved in all the processes in the island.

See the section below on Infrastructure repository for additional communications overhead.

**Infrastructure Repository**
There is no infrastructure database required for this type of clustering - each 9iAS instance is a stand-alone instance not requiring any support from an infrastructure repository. However, each instance can be configured to have an infrastructure repository, or in the more common case, they will all share a common repository.

In case all systems are indeed part of a farm (i.e. share the same repository), the notifications are sent to all the instances in the farm. However, in the current scenario these notifications will be dropped since the instances are all independent.

Sharing a repository across all instances may be perceived as introducing a single point of failure. However 9iAS (core Install) does not use the repository during steady state functioning of the web site and hence the impact of a repository downtime is limited only to management functions such as application deployment, joining/removing clusters etc.

**Summary of Loosely Bound cluster**
This deployment option was available in the previous releases too, and is widely used since it is very simple to setup. While it does provide the new fault tolerance features, it does not provide the benefits of distributed configuration and load balancing (at mod_oc4j level), which are introduced new in Release 2.

The next deployment scenario – the 9iAS J2EE Web Application Cluster – resolves these shortcomings.
THE J2EE WEB APPLICATION CLUSTER:
This deployment scenario is very similar to the one discussed in the previous section, with two primary differences:

- The 9iAS instances are part of a 9iAS cluster, and
- The installation of the 9iAS instances is of type core - no other install types are cluster-able, and hence the discussion in this section does not apply. For those cases the loosely coupled cluster (or its associated variations) have to be used.

Description
In this scenario, a 3rd party load balancer or 9iAS Web Cache instance, routes to 9iAS instances at the back-end that have been defined to be part of a 9iAS cluster. Hence, they are identical in almost all regards. Both scenarios do not spell out EJB clustering – and the concepts discussed in the EJB cluster chapter remain applicable.

There are three major capabilities that this deployment provides over the previously discussed loose clustering:

1. The OHS in any instance in the cluster can route to an OC4J in any instance in the cluster.
2. The 9iAS clustering infrastructure (OPMN, DCM) know of all the instances, and hence can manage them as described in the clustering chapter - single click application deployment across instances, and, update routing table(s) if a process in the cluster is down.
3. Enable islands that are smaller than cluster size, for more efficient session replication.

The load balancer in the front is however configured to load balance requests

\[\text{For correctness - it is also possible to enable islands that are smaller than the cluster size in loosely coupled clusters. However, the manageability gets difficult, since now some members of the cluster have a different configuration than others.}\]
across all nodes. These nodes will need to be running the same operating system. Notice that the front-end load balancer – whether Web Cache, or an external load balancer – does not need session stickiness. They can continue to route to any OHS, as though every request they receive from the browser is stateless. When the request gets to OHS, as discussed earlier, it interprets the cookie and routes it to the right OC4J process.

With this description, we will now cover the clustering characteristics of this J2EE cluster.

**J2EE Web Applications Clustering Characteristics**

**Availability: Transparent Failover**

In case of failure of the back-end system, requests with no existing sessions are, as before, routed to an alternate 9iAS instance without any impact. If the request is associated with a session and:

1. The OC4J process that handled the request previously is down - In this case, mod_oc4j simply routes to the next OC4J process in the same island - even if that process is on a different 9iAS instance. This routing was not possible in the loosely bound cluster case, and the work around there was to have two OC4J processes in the same island in an instance.

2. The entire node (or just OHS) is down - The external load balancer / Web Cache, will simply route the request to an alternate node. And session replication and island processes will ensure a correct response.

3. Failure During a Request continues to be visible to the end-user. However, if he does re-request (assuming it is a non-POST request and safe to repeat), the request simply gets treated as one of the above two cases.

**Availability: Quick Automated Restarts**

There is no change here - the system continues to provide the quick automated restarts of down processes without requiring any administrative intervention.

In addition though, the system now communicates to other instances in the cluster ensuring everyone is aware of the global state of the up/down processes in the cluster.

**Availability: Shared “Cluster” State**

As before, the session is replicated across the processes in the island.

In addition, the routing table is also considered a shared state in this kind of deployment architecture. The routing table, as discussed in the chapter on

---

7 In 9.0.2, heterogeneous operating system clusters are certified for Web Cache clusters. For the rest of 9iAS components in core install, such clustering may work, but is not certified.
clustering framework, is updated if new processes join or old ones die. This ensures mod_oc4j always routes to a process that is up.

**Availability: Graceful Degradation**

In earlier case, a down OC4J process could result in failure visible to the end-user, but it provided graceful degradation in the case of a node failure.

In this clustering setup, an OC4J process failure is better masked due to the routing capabilities of mod_oc4j across all instances in the cluster. As before, this degradation is only temporary, since the new clustering infrastructure in 9iAS will automatically restart the down instances.

**Availability: Any Single Points of Failure**

The discussion in the loosely coupled architecture case applies here too. Specifically, if configured right, no 9iAS component is a single point of failure in this deployment scenario.

**Scalability: Load Balancing**

Again, majority of the previous discussion is applicable here too. The primary difference is in the case of J2EE applications: OHS with mod_oc4j is now able to select from a larger number of processes to route the request to. In the loosely coupled architecture case, mod_oc4j could choose processes only from within an island on the current instance, whereas now it can choose any OC4J process from the island across all instances.

Also, Web Cache and the external load balancer do not need session stickiness, since mod_oc4j ensures the request gets routed to the right OC4J.

**Scalability: Distributed Caching**

There is no difference in system capabilities from the loosely coupled clustering - Web Cache and Java Object Cache continue to provide distributed clustering capabilities.

**Manageability: Clustered and Rolling Deployment**

This is an area of major change from loosely coupled cluster – now applications may be deployed across all instances in the cluster, with a single click deployment mechanism from OEM.

This saves the trouble of manually ensuring all systems in the cluster are running the correct version of a certain application.

The system does not directly provide rolling deployment. However, it can be easily simulated by moving a machine out of a cluster and then having it join a cluster with updated configuration – just as discussed in the loosely coupled deployment scenario. The only difference is in the process to join and leave a
Manageability: Cluster Re-Configuration

Joining a Cluster: The only pre-requisite to joining an existing cluster is to have a 9iAS core install and share the same repository. As long as this is the case, it can join a cluster with ease – with a single button click in OEM. The new cluster member will automatically get all the same J2EE configuration as the rest of the machines in the cluster. All OHS instances (i.e. mod_oc4j) in the cluster will begin routing the requests to it.

If you would like the OHS on this new node to receive HTTP requests from the external load balancer (or Web Cache) the router (or Web Cache) has to be configured to route to the new cluster member.

Removing from cluster:

1. Remove the machine from the load balancer configuration, so that OHS does not receive requests.

2. Remove the instance from the 9iAS cluster, so that none of the OC4J processes in this instance receive requests from other OHS in the cluster.

Clustering Overhead: Security

DCM on every node / instance needs to talk to a repository for configuration and deployment information. If a database is used as a repository, additional firewall configuration may be required to enable DCM to talk to this infrastructure repository through firewalls.

If instances within a cluster span sub-nets, additional configuration will be required to enable OPMN events / notifications to cross the routers / firewalls between those. It is not a good idea to let processes in an island cross sub-nets.

The OPMN events are http, and not https and hence it is recommended that the entire zone where these messages are transmitted should be behind a firewall. Moreover, the specific remote ports used for OPMN communication should be blocked to prevent someone external from either viewing the cluster state through an OPMN query, or, modifying the cluster state by giving a false notification.

Clustering Overhead: Communications

OPMN communication is point to point. As the number of members in a farm (independent of their cluster membership status) increases, the communication overhead due to OPMN events increases too (O (N**2) ). However, the messages are small and do not noticeably increase the network traffic.

This deployment scenario gives the ability to have an island size smaller than the cluster size. This reduces the memory overhead introduced by session replication resulting in better performance tuning.
Other than that, there is no difference from a performance perspective, between the loose cluster setup and this scenario.

**Infrastructure Repository**

An infrastructure repository makes it easier to store the entire configuration in one place and thus enables the single click deployment, addition of new nodes to a cluster with no manual configuration etc.

As mentioned in the earlier section, a shared repository does not translate into a single point of failure. Its downtime impacts only the management operations of the site – such as deploying new applications, joining / removing nodes from a cluster etc.

It is also possible to use a file-based repository instead of a database-backed repository. However, not all capabilities are available in this scenario. A useful setup is discussed in the loosely coupled cluster with manual islands and routing scenario.

**Summary of a J2EE Web Application Cluster**

This is the recommended way to deploy applications for 9i AS Release 2.

It provides single click deployment across the cluster, automated failover and re-routing of requests, and frameworks that enable distribution of application java objects. Checkpointing capability for cluster and instance configuration provides better backup and recovery in case of catastrophic crash. And automated restarts minimize the administrator intervention, making Oracle9i AS cluster a self-managed system.

This setup provides the best availability, scalability, and manageability of an Oracle9i AS Cluster.

**LOOSELY COUPLED CLUSTER WITH MANUAL ROUTING**

This variation of the loosely coupled cluster is specifically designed to alleviate the drawback that an OC4J failure may be visible to the end user.

In this scenario the mod_oc4j component of OHS is manually configured to make it aware of other OC4J processes running in the loosely coupled cluster.

Each of the independent instances can have processes that are part of the same island. This provides added failover capability in case a node goes down.
With this manual update, mod_oc4j now receives event notifications about the state of all OC4J in the loosely coupled cluster, thus, enabling it to route to those.

With this addition, all of the dynamic routing and notification capabilities of mod_oc4j and OPMN are now available. However, DCM capabilities (distributed configuration) are still not available, since these independent instances are still not part of the same cluster.

This setup may be configured either with or without an infrastructure repository. However, the manual configuration is a little easier if an infrastructure repository is present.

Since this setup is a variation of what we discussed earlier, we do not go into the detailed clustering analysis for this setup.

**9iAS Clusters Across Firewalls**

In this scenario, there may be a cluster of 9iAS instances composed of only OHS processes – which may be deployed in one zone in a DMZ. Behind the second firewall may be yet another cluster of 9iAS instances, composed mainly of OC4J instances, which may further be broken down in to islands.

The J2EE web application cluster discussion from the earlier section applies to both the clusters individually – including single click management and fault tolerance.

However, some manual configuration is required to ensure that analysis is applicable to the system consisting of two individual clusters. In particular:

1. If the two clusters share the same repository, mod_oc4j in the OHS cluster needs to be configured to route requests to the backend cluster – a simple single line update in the configuration file.

2. If the two clusters do not share the repository, additional configuration is required to notify mod_oc4j of each instance running in the backend.

3. Some firewall configuration also may be required to ensure mod_oc4j and OPMN communication goes through the firewall.

From an availability stand-point, if an OHS dies, the external load balancer re-routes. If an OC4J dies, mod_oc4j in any OHS automatically re-routes to another
process in the same island.

From a scalability perspective, mod_oc4j load balances requests across all Oc4J cluster.

From a manageability perspective, the individual clusters can be managed with the single click deployment and management through OEM. However, certain actions - such as J2EE web application deployments that require updating the root context in OHS - will need manual steps.

OTHER 9iAS COMPONENTS

This document covers only 9iAS Core Installation. 9iAS Release 2 contains significant additional capabilities such as portal, business intelligence, wireless etc. This document has not touched clustering of those components. The individual component guides contain the specifics of clustering of those components.

Moreover, just to repeat a point made earlier in the doc, the core clustering framework and its configuration through OEM becomes ineffective for installations that contain components beyond core.

Consider this situation: There are two nodes with a portal installation of 9iAS, and there are two applications running - a portal and a J2EE web application on both the nodes. The J2EE web application can then not be clustered through OEM and DCM as described in this document.

Thus the implication to be aware of is that manual steps are required to enable clustering when using more than the core components of 9iAS.

ORACLE9iAS CLUSTERING SUMMARY

This paper discussed the typical characteristics of a successful clustering framework, and applied the stringent evaluation criteria to the individual components and to a couple of sample deployment architectures of 9iAS.

In summary, Oracle9iAS Release 2 provides enables single click distributed deployment of J2EE applications. It monitors and restarts processes in a cluster and transparently updates the routing table for optimal load balancing and minimum runtime impact on requests, without requiring any administrator intervention.

These enhancements are expected to make the total cost of ownership of the product even lower, while enhancing the availability, scalability, and manageability of deployments using Oracle9iAS Release 2.