Oracle9*i* Real Application Clusters Concepts Release 1 (9.0.1)
Part Number A89867-02

Home Book List Contents Index Master Index Feedback

Previous Next

10 High Availability Concepts and Best Practices

This chapter describes the concepts and some of the *best practices* methodologies for use in Real Application Clusters to implement high availability. This chapter includes the following topics:

Understanding High Availability

Computing environments configured to provide nearly full-time availability are known as **high availability** systems. Such systems typically have redundant hardware and software that makes the system available despite failures. Well-designed high availability systems avoid having single points-of-failure. Any hardware or software component that can fail has a redundant component of the same type.

When failures occur, the **failover** process moves processing performed by the failed component to the backup component. This process remasters systemwide resources, recovers partial or failed transactions, and restores the system to normal, preferably within a matter of microseconds. The more transparent that failover is to users, the higher the availability of the system.

Oracle has a number of products and features that provide high availability. These include Real Application Clusters, **Oracle Real Application Clusters Guard**, Oracle Replication, and **Oracle9i Data Guard**. These can be used in various combinations to meet your specific high availability needs. This chapter describes high availability within the context of Real Application Clusters.

Real Application Clusters are inherently high availability systems. Clusters typical of Real Application Clusters environments, as described in Chapter 3, can provide continuous service for both planned and unplanned outages.

Note:

More on the topic of High Availability is included in Chapter 11 and Chapter 12. These chapters describe high availability within the context of Oracle Real Application Clusters Guard.

Measuring Availability

You can classify systems and evaluate their expected availability by system type. Mission-critical and business-critical applications such as e-mail and Internet servers probably require a significantly greater availability than do applications that have a smaller number of users. As well, some systems could have a *continuous* (24 hours a day, seven days a week) uptime requirement, while others such as a stock market tracking system will have nearly continuous uptime requirements for specific time frames, such as when a stock market is open.

High Availability Measurements

The software industry generally measures availability by using two types of metrics (measurements):

- The mean time to recover
- The mean time between failures

For most failure scenarios, the industry focuses on **mean time to recover (MTTR)** issues and investigates how to optimally design systems to reduce these. MTBF is generally more applicable to hardware availability metrics; this chapter does not go into detail about **mean time between failures (MTBF)**. However, given that you can design Real Application Clusters to avoid single points-of-failure, component failures might not necessarily result in application unavailability. Hence, Real Application Clusters can greatly reduce the MTBF from an application availability standpoint.

Another metric that is generally used is *number of nines*. For example, 526 minutes of system unavailability for each year results in 99.9% or **three nines availability**.

Five minutes of system unavailability for each year results in a 99.999% or **five nines availability**.

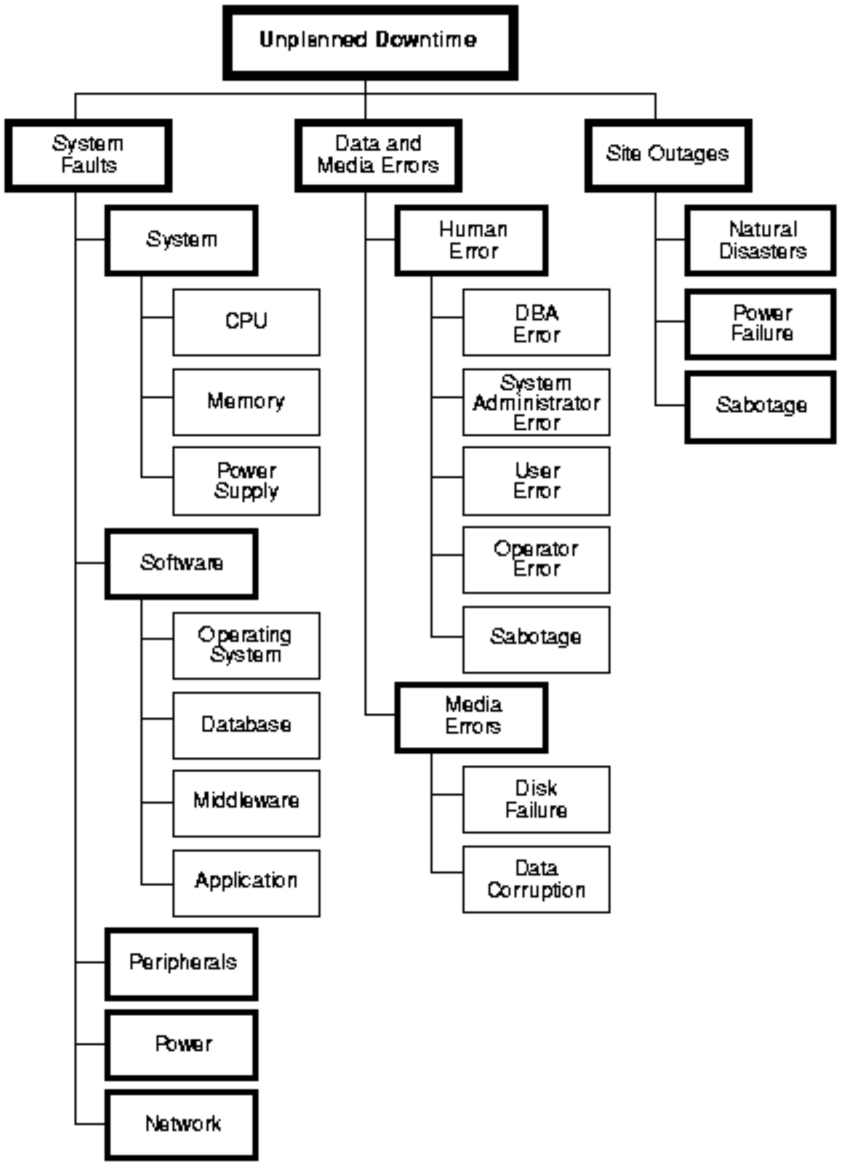
It is difficult to consider several nines of availability without also describing the ground rules and strict processes for managing application environments, testing methodologies, and change management procedures. For these reasons, Real Application Clusters can significantly reduce MTTR during failures. This inevitably contributes toward a more favorable availability for an entire system.

As mentioned, a well designed Real Application Clusters system has redundant components that protect against most failures and that provide an environment without single points-of-failure. Working with your hardware vendor is key to building fully redundant cluster environments for Real Application Clusters.

Causes of Downtime

Downtime can be either planned or unplanned. Figure 10-1 shows many causes of **unplanned downtime**. They can be broadly classified as system faults, data and media errors, and site outages. This unplanned downtime is disruptive because it is difficult to predict its timing.

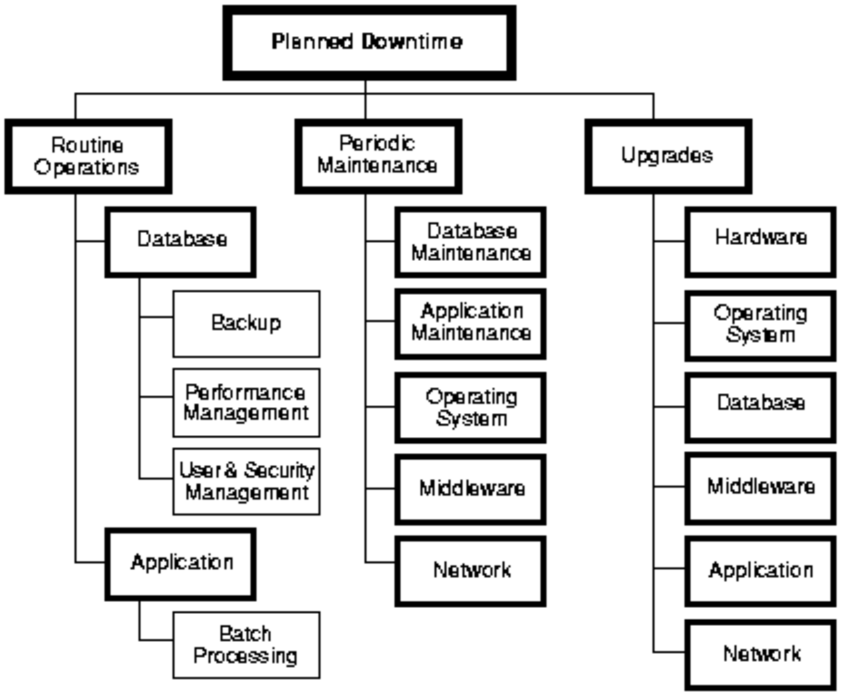
Figure 10-1 Causes of Unplanned Downtime



Text description of the illustration pfscn012.gif

However, **planned downtime** can be just as disruptive to operations, especially in global enterprises that support users in multiple time zones. Figure 10-2 shows several kinds of planned downtime. They can be generally classified as routine operations, periodic maintenance, and upgrades.

Figure 10-2 Causes of Planned Downtime



Text description of the illustration pfscn013.gif

The key to building a high availability solution is to consider all these causes of downtime and to design a solution that addresses them. Oracle Real Application Clusters Guard is Oracle's premier high availability solution. It has been designed to address many of the causes of unplanned and planned downtime, to maximize database availability, and to minimize disruption to end users of the data services.

Planning for High Availability

High availability is the result of thorough planning and careful system design. You can conduct high availability planning at two levels:

- The system level with a broad perspective
- The failure protection level to ensure against a long list of potential causes of failures

System Level Planning

System level planning involves:

- Capacity Planning
- Redundancy Planning

Capacity Planning

High availability requires the timely processing of transactions for a system to be deemed completely *available*. While this chapter does not provide extended capacity planning discussions, adequate system resources for managing application growth are important for maintaining availability.

If an application runs on a single **Symmetric Multi-Processor (SMP)** machine with single **instance** Oracle, a natural growth path is to migrate this database to a larger SMP machine. However, depending on your hardware vendor's product line, this might not be an option. Migrating to Real Application Clusters, however, might be an option.

Real Application Clusters enables you to add nodes to your system to increase capacity and handle application growth. You can do this with minimal interference to existing client transactions.

Redundancy Planning

Redundancy planning means duplicating system components so that no single component failure increases system downtime. Redundant components are often used in high-end SMP machines to protect against failures. For example, redundant power supplies and redundant cooling fans are not uncommon in high-end SMP server systems. A clustered Real Application Clusters environment can extend this redundant architecture by creating complete redundancy so that there is no single point-of-failure.

Note:

When installing a high availability system, ensure the hardware and software are certified as a unit.

Configuring Real Application Clusters for High Availability

Real Application Clusters builds higher levels of availability on top of the standard Oracle features. All single instance high availability features such as Fast-Start Recovery and online reorganizations apply to Real Application Clusters as well. Fast-Start Recovery can greatly reduce MTTR with minimal effects on online application performance. Online reorganizations reduce the durations of planned downtimes. Many operations can be performed online while users update the underlying objects. Real Application Clusters preserves all these standard Oracle features.

In addition to all the regular Oracle features, Real Application Clusters exploits the redundancy provided by clustering to deliver availability with *n*-1 node failures in an *n*-node cluster. In other words, all users have access to all data as long as there is one available node in the cluster.

To configure Real Application Clusters for high availability, you must carefully consider the hardware and software component issues of your cluster as described the following section.

Cluster Components and High Availability

This section describes high availability and cluster components in the following sections:

- Cluster Nodes
- Cluster Interconnects
- Storage Devices
- Operating System Software and Cluster Managers
- Database Software

See Also:

Chapter 3 for more information about these components

Cluster Nodes

Real Application Clusters environments are fully redundant because all nodes access all the disks in the cluster. The failure of one node does not affect another node's ability to process transactions. As long as the cluster has one surviving node, all database clients can process all transactions, although subject to increased response times due to capacity constraints on the one node.

Cluster Interconnects

Interconnect redundancy is often overlooked in clustered systems. This is because the **mean time to failure (MTTF)** is generally several years.

Therefore, cluster interconnect redundancy might not be a high priority. Also, depending on the system and sophistication level, a redundant cluster interconnect could be cost prohibitive and have insufficient business justification.

However, a redundant cluster interconnect is an important aspect of a fully redundant cluster. Without this, a system is not truly free of single points-of-failure. Cluster interconnects can fail for a variety of reasons and not all of them are accounted for. Nor can the be accounted for when manufacturer MTTF metrics are provided. Interconnects can fail due to either device malfunctions, such as an oscillator failure in a switch interconnect, or because of human error.

Storage Devices

Real Application Clusters operate on a single image of the data; all nodes in the cluster access the same set of data files. Database administrators are encouraged to use hardware-based mirroring to maintain redundant media. In this regard, Real Application Clusters are no different from single instance Oracle. Disk redundancy depends on the underlying hardware and software mirroring in use, such as a **Redundant Array of Independent Disks (RAID)**.

Operating System Software and Cluster Managers

Real Application Clusters environments have full node redundancy; each node runs its own operating system copy. Hence, the same considerations about node redundancy also apply to the operating system. The **Cluster Manager (CM)** is an extension of the operating system. Since the Cluster Manager software is also installed on all the nodes of the cluster, full redundancy is assured.

Database Software

In Real Application Clusters, Oracle executables (such as Oracle home) are installed on the local disks of each node and an instance runs on each node of the cluster. Note that if your supports a cluster file system (CFS) then only one copy of Oracle home will be installed. All instances have equal access to all data and can process any transactions. In this way, Real Application Clusters ensure full database software redundancy.

Disaster Planning

Real Application Clusters are primarily a single site, high availability solution. This means the nodes in the cluster generally exist within the same building, if not the same room. Thus, disaster planning can be critical. Disaster planning covers planning for fires, floods, hurricanes, earthquakes, terrorism, and so on. Depending on how mission critical your system is, and the propensity of your system's location for such disasters, disaster planning could be an important high availability component.

Oracle offers other solutions such as Oracle9*i* Data Guard and Oracle Replication to facilitate more comprehensive disaster recovery planning. You can use these solutions with Real Application Clusters where one cluster hosts the primary database and another remote system or cluster hosts the disaster recovery database. However, Real Application Clusters are not required on either site for purposes of disaster recovery.

Failure Protection Validation

Once you have carefully considered your system level issues, validate that the Real Application Clusters environment optimally protects against potential failures. Use the following list of failure causes to plan and troubleshoot your failure protection system:

- Cluster component
- CPU
- Memory
- Interconnect software
- Operating System
- Cluster Manager
- Oracle database instance media
- Corrupt or lost control file
- Corrupt or lost log file
- Corrupt or lost data file
- Human error
- Dropped or deleted database object

Real Application Clusters environments protect against cluster component failures and software failures. However, media failures and human error could still cause system downtime. Real Application Clusters, as with single instance Oracle, operates on one set of files. For this reason, you should adopt best practices to avoid the adverse effects of media failures.

RAID-based redundancy practices avoid file loss but might not prevent rare cases of file corruptions. Also, if you mistakenly drop a database object in an Real Application Clusters environment, then you can recover that object the same way you would in a single instance database. These are the primary limitations in an otherwise very robust and highly available Real Application Clusters system.

Once you deploy your system, the key issue is the transparency of failover and its duration.

See Also:

"System Level Planning" for more information about how Real Application Clusters environments protect against cluster component failures and software failures

Failover and Real Application Clusters

This section describes the basic principles of failover and the various features Real Application Clusters offer to implement it in high availability systems. Topics in this section include:

- Failover Basics
- Client Failover
- Uses of Transparent Application Failover
- Server Failover

Failover Basics

Failover requires that highly available systems have accurate instance monitoring or **heartbeat** mechanisms. In addition to having this functionality for normal operations, the system must be able to quickly and accurately synchronize resources during failover.

The process of synchronizing, or *remastering*, requires the graceful shutdown of the failing system as well as an accurate assumption of control of the resources that were mastered on that system. Accurate remastering also requires that the system have adequate information about resources across the cluster. This means your system must record resource information to remote nodes as well as local. This makes the information needed for failover and recovery available to the recovering instances.

See Also:

- *Oracle Real Application Clusters Guard Administration and Reference Guide* for information how to set up Oracle Real Application Clusters Guard on your system
- *Oracle9i Recovery Manager User's Guide* for details on recovery

Duration of Failover

The duration of failover includes the time a system requires to remaster systemwide resources and recover from failures. The duration of the failover process can be a relatively short interval on certified platforms.

- For existing users, failover entails both server and client failover actions.
- For new users, failover only entails the duration of server failover processing.

Client Failover

It is important to hide system failures from database client connections. Such connections can include application users in client server environments or middle-tier database clients in multitiered application environments. Properly configured failover mechanisms transparently reroute client sessions to an available node in the cluster. This capability in the Oracle database is referred to as Transparent Application Failover.

Transparent Application Failover

Transparent Application Failover (TAF) enables an application user to automatically reconnect to a database if the connection fails. Active transactions roll back, but the new database connection, made by way of a different node, is identical to the original. This is true regardless of how the connection fails.

How Transparent Application Failover Works

With Transparent Application Failover, a client notices no loss of connection as long as there is one instance left serving the application. The **database administrator (DBA)** controls which applications run on which instances and also creates a failover order for each application.

Elements Affected by Transparent Application Failover

During normal client/server database operations, the client maintains a connection to the database so the client and server can communicate. If the server fails, so then does the connection. The next time the client tries to use the connection the client issues an error. At this point, the user must log in to the database again.

With Transparent Application Failover, however, Oracle automatically obtains a new connection to the database. This enables users to continue working as if the original connection had never failed.

There are several elements associated with active database connections. These include:

- Client/Server database connections
- Users' database sessions executing commands
- Open cursors used for fetching
- Active transactions
- Server-side program variables

Transparent Application Failover automatically restores some of these elements. However, you might need to embed other elements in the application code to enable transparent application failover.

See Also:

Oracle9i Net Services Administrator's Guide for background and configuration information on Transparent Application Failover

Uses of Transparent Application Failover

While the ability to fail over client sessions is an important benefit of Transparent Application Failover, there are other useful scenarios where Transparent Application Failover improves system availability. These topics are discussed in the following subsections:

- Transactional Shutdowns
- Quiescing the Database
- Load Balancing
- Transparent Application Failover Restrictions
- Database Client Processing During Failover

Transactional Shutdowns

It is sometimes necessary to take nodes out of service for maintenance or repair. For example, if you want to apply patch releases without interrupting service to application clients. Transactional shutdowns facilitate shutting down selected nodes rather than an entire database. Two transactional shutdown options are available:

- Use the `TRANSACTIONAL` clause of the `SHUTDOWN` statement to take a node out of service so that the shutdown event is deferred until all existing transactions are completed. In this way client sessions can be migrated to another node of the cluster at transaction boundaries.
- Use the `TRANSACTIONAL LOCAL` clause of the `SHUTDOWN` statement to do a transactional shutdown on a specified local instance. This command can be used to prevent new transactions from starting locally, and to perform an immediate shutdown after all local transactions have completed. With this option, you can gracefully move all sessions from one instance to another by shutting down selected instances transactionally.

After performing a transactional shutdown, newly submitted transactions get routed to an alternate node in the cluster. An immediate shutdown is performed on the node when all existing transactions complete.

Quiescing the Database

You may need to perform administrative tasks that require isolation from concurrent user transactions or queries. To do this, you can use the quiesce database feature. This prevents you, for example, from having to shut down the database and re-open it in restricted mode to perform such tasks.

To do this, you can use the `ALTER SYSTEM` statement with the `QUIESCE RESTRICTED` clause.

Note:

You cannot open the database on one instance if the database is *being quiesced* on another node. In other words, if you issued the `ALTER SYSTEM QUIESCE RESTRICTED` statement but it is not finished processing, you cannot open the database. Nor can you open the database if it is already in a quiesced state.

See Also:

The *Oracle9i Database Administrator's Guide* for more detailed information on the quiesce database feature and the *Oracle9i SQL Reference* for more information about the `ALTER SYSTEM QUIESCE RESTRICTED` syntax

The QUIESCE RESTRICTED clause enables you to perform administrative tasks in isolation from concurrent user transactions or queries.

See Also:

Oracle9i Real Application Clusters Administration for information on the Quiesce Database feature

Load Balancing

A database is available when it processes transactions in a timely manner. When the load exceeds a node's capacity, client transaction response times are adversely affected and the database availability is compromised. It then becomes important to manually migrate a group of client sessions to a less heavily loaded node to maintain response times and application availability.

In Real Application Clusters, the Transport Network Services (TNS) listener provides automated load balancing across nodes in both shared server and dedicated server configurations.

Connection load balancing improves connection performance by balancing the number of active connections among multiple dispatchers. In a single-instance environment, the listener selects the least loaded dispatcher to handle the incoming client requests. In a Real Application Clusters environment, connection load balancing also has the capability to balance the number of active connections among multiple instances.

Due to dynamic service registration, a listener is always aware of all of the instances and dispatchers regardless of their locations. Depending on the load information, a listener decides which instance and, if in shared server configuration, to which dispatcher to send the incoming client request.

In a shared server configuration, a listener selects a dispatcher in the following order:

1. Least loaded node
2. Least loaded instance
3. Least loaded dispatcher for that instance

In a dedicated server configuration, a listener selects an instance in the following order:

1. Least loaded node
2. Least loaded instance

If a database service has multiple instances on multiple nodes, then the listener chooses the least loaded instance on the least loaded node. If shared server is configured, then the least loaded dispatcher of the selected instance is chosen.

.

See Also:

Oracle9i Net Services Administrator's Guide for more information on Load Balancing

Transparent Application Failover Restrictions

When a connection is lost, you might experience the following:

- All PL/SQL package states on the server are lost at failover
- ALTER SESSION statements are lost

- If failover occurs when a transaction is in process, then each subsequent call causes an error message until the user issues an OCITransRollback **call**. Then Oracle issues an **Oracle Call Interface (OCI)** success message. Be sure to check this message to see if you must perform additional operations.
- Continuing work on failed-over cursors can cause an error message

If the first command after failover is not a SQL SELECT or OCIStmtFetch statement, then an error message results. Failover only takes effect if the application is programmed with OCI release 8.0 or greater.

Database Client Processing During Failover

Failover processing for query clients is different than the failover processing for Database Mount Lock clients. The important issue during failover operations in either case is the failure is masked from existing client connections as much as possible. The following subsections describe both types of failover processing.

Query Clients

At failover, in-progress queries are reissued and processed from the beginning. This might extend the duration of the next query if the original query took a long time. With Transparent Application Failover (TAF), the failure is masked for query clients with an increased response time being the only client observation. If the client query can be satisfied with data in the buffer cache of the surviving node that the client reconnected to, then the increased response time is minimal. By using TAF's PRECONNECT method eliminates the need to reconnect to a surviving instance and thus further minimizes response time. However, `PRECONNECT` allocates resources awaiting the failover event.

If the client query cannot be satisfied with data in the buffer cache of the reconnect node, then disk I/O is necessary to process the client query. However, server-side recovery needs to complete before access to the data files is allowed. The client transaction experiences a system pause until server-side recovery completes, if server-side recovery has not already completed.

You can also use a callback function through an OCI call to notify clients of the failover so that the clients do not misinterpret the delay for a failure. This prevents the clients from manually attempting to reestablish their connections.

Database Mount Lock Clients

Database Mount Lock database clients perform INSERT, UPDATE, and DELETE operations. In-progress Database Mount Lock transactions on a failed instance may restart on a surviving instance without client knowledge providing the application code fully exploits the OCI libraries. This achieves application failover, without manual reconnects, but requires application-level coding. The required code handles certain Oracle errors and performs a reconnect when those errors occur.

Without this application code, INSERT, UPDATE, and DELETE operations on the failed instance return an un-handled Oracle error code. In such cases, Oracle resubmits the transaction for execution. Upon re-submission, Oracle routes the client connections to a surviving instance. The client transaction then stops only momentarily until server-side recovery completes.

Server Failover

Server-side failover processing in Real Application Clusters is different from host-based failover solutions that are available on many server platforms. The following subsections describe both types of failover processing.

Host-Based Failover

Many operating system vendors and other cluster software vendors offer high availability application failover products. These failover solutions monitor application services on a given primary cluster node. They then fail over such services to a secondary cluster node as needed. Host-based failover solutions generally have one active instance performing useful work for a given database application. The secondary node monitors the application service on the primary node and initiates failover when the primary node service is unavailable.

Failover in host-based systems usually includes the following steps.

1. Detecting failure by monitoring the heartbeat
2. Reorganizing cluster membership in the Cluster Manager
3. Transferring disk ownership from the primary node to a secondary node
4. Restarting application and database binaries (Oracle executables)
5. Performing application and database recovery
6. Reestablishing client connections to the failover node

Real Application Clusters Failover

Real Application Clusters provide rapid server-side failover. This is accomplished by the concurrent, active-active architecture in Real Application Clusters. In other words, multiple Oracle instances are concurrently active on multiple nodes and these instances synchronize access to the same database. All nodes also have concurrent ownership and access to all disks. When one node fails, all other nodes in the cluster maintain access to all the disks; there is no disk ownership to transfer, and database application binaries are already loaded into memory.

Depending on the size of the database, the duration of failover can vary. The larger the database, or the greater the size of its data files, the greater the failover benefit in using Real Application Clusters.

How Failover Works

The following subsections describe server failover recovery processing in Real Application Clusters:

- Detecting Failure
- Reorganizing Cluster Membership
- Performing Database Recovery

Detecting Failure

Real Application Clusters relies on the Cluster Manager software for failure detection because the Cluster Manager maintains the heartbeat functions. The time it takes for the Cluster Manager to detect that a node is no longer in operation is a function of a configurable heartbeat timeout parameter. This parameter varies, depending on your platform. You can configure this value on most systems. Defaults can vary significantly, depending on your clusterware. (Such as Sun Cluster or Hewlett-Packard Service Guard OPS Edition.) The parameter value is inversely related to the number of false failure detections because the cluster might incorrectly determine that a node is failing due to transient failures if the timeout interval is set too low. When a failure is detected, cluster reorganization occurs.

Reorganizing Cluster Membership

When a node fails, Oracle must alter the node's cluster membership status. This is known as a *cluster reorganization* and it usually happens quickly. The duration of cluster reorganization is proportional to the number of surviving nodes in the cluster.

The **Global Cache Service (GCS)** and **Global Enqueue Service (GES)** provide the Cluster Manager interfaces to the software and expose the cluster membership map to the Oracle instances when nodes are added or deleted from the cluster. The LMON process on each cluster node communicates with the Cluster Manager on the respective nodes and exposes that information to the respective Oracle instances.

LMON also provides another useful function by continually sending messages from the node it runs on and often writing to the shared disk. The absence of these functions from any node is evidence to the surviving nodes that the node is no longer a member of the cluster. Such a failure causes a change in a node's membership status within the cluster and LMON initiates the recovery actions that include remastering of Global Cache Service and Global Enqueue Service resources and instance recovery.

At this stage, the Real Application Clusters environment is in a state of system pause, and most client transactions suspend until Oracle completes recovery processing.

Note: :

LMON-provided services are also referred to as **cluster group services (CGS)**

Instance Membership Recovery

The process of **instance membership recovery (IMR)** guarantees that all members of a cluster are functional (**active**) The IMR process does the following:

- Ensures that communications are viable between all members, and that all members are capable of responding.
- Provides a mechanism for removing members that are not active. IMR arbitrates the membership and removes members that it decides no longer belong to the cluster.
- Votes on membership using the Control File. Each member writes a bitmap to the Control File. This is part of the checkpoint progress record.
- Removes members based on a communication failure. IMR will perceive members as *expired* if they do not provide their normal periodic heartbeat

to the Control File, or if they do not respond to a query on their status.

- Settles the membership votes and locks the **control file voting results record (CFVRR)** with an arbiter.

All instances read the CFVRR. If a member is not in the membership map, then IMR assumes a node has expired. Appropriate diagnostic information is provided. As IMR is currently configured, all members wait indefinitely for notification of node expiration. There is no forced removal of instances.

Part of the fault tolerance of Real Application Clusters is provision for the possibility that the IMR arbiter itself could fail.

Performing Database Recovery

When an instance fails, Oracle must remaster Global Cache Service resources from the failed instance onto the surviving cluster nodes. The database recovery process in Real Application Clusters includes these topics discussed in the following sections:

- Remastering Global Cache Service Resources of the Failed Instance
- Instance Recovery

Remastering Global Cache Service Resources of the Failed Instance

The time required for remastering locks is proportional to the number of Global Cache Service resources in the failed instance. This number in turn depends upon the size of the buffer caches. These processes occur in parallel.

During this phase, all resources previously mastered at the failed instance are redistributed across the remaining instances. These resources are reconstructed at their new master instance. All other resources previously mastered at surviving instances are not affected. For any lock request, there is a 1/*n* chance that the request will be satisfied locally and a (*n*-1)/*n* chance that the lock request involves remote operations. In the case of one surviving instance, all lock operations are satisfied locally.

Once remastering of the failed instance Global Cache Service resource is completed, Oracle must clean up the in-progress transactions of the failed instance. This is known as *instance recovery*.

Instance Recovery

Instance recovery includes cache recovery and transaction recovery. Instance recovery requires that an active Real Application Clusters instance detects failure and performs recovery processing for the failed instance. The first Real Application Clusters instance that detects the failure, by way of its LMON process, controls the recovery of the failed instance by taking over its redo log files and performing instance recovery. This is why the redo log files must be on a shared device such as a shared raw logical volume or cluster file system.

Instance recovery is complete when Oracle has performed the following steps:

- Rolling back all uncommitted transactions of the failed instance. This is also known as *transaction recovery*.
- Replaying the online redo log files of the failed instance.

Since Oracle can perform transaction recovery in a deferred fashion, client transactions can begin processing when cache recovery is complete.

You do not need to recover entire files in the event of a failure. You can recover individual blocks.

.

See Also:

Oracle9i Recovery Manager User's Guide and Reference for a description of Block Media Recovery (BMR)

Cache Recovery

For cache recovery, Oracle replays the online redo logs of the failed instance. Oracle performs cache recovery using parallel execution so that parallel processes, or threads, replay the redo logs of the failed Oracle instance. It could be important that you keep the time interval for redo log replay to a predictable duration. The Fast-Start Recovery feature in Oracle9*i* enables you to control this.

Oracle also provides nonblocking rollback capabilities. This means that full database access can begin as soon as Oracle has replayed the online log files. After cache recovery completes, Oracle begins transaction recovery.

See Also:

Oracle9i Database Performance Guide and Reference for more information on how to use Fast-Start Recovery

Transaction Recovery

Transaction recovery comprises rolling back all uncommitted transactions of the failed instance. Uncommitted transactions are *in-progress* transactions that did not commit.

The Oracle9*i* Fast-Start Rollback feature performs this as deferred processing that runs in the background. Oracle uses a multi-version read consistency technology to provide on-demand rollback of only those rows blocked by expired transactions. This allows new transactions to progress with minimal delay. New transactions do not have to wait for long-running expired transactions to be rolled back. Therefore, large transactions generally do not affect database recovery time.

Just as with cache recovery, Oracle9*i* Fast-Start Rollback rolls back expired transactions in parallel. However, single instance Oracle rolls back expired transactions by using the CPU of one node.

Real Application Clusters provide cluster-aware Fast-Start Rollback capabilities that use all the CPU nodes of a cluster to perform parallel rollback operations. Each cluster node spawns a recovery coordinator and recovery processes to assist with parallel rollback operations. The Fast-Start Rollback feature is thus *cluster aware* because the database is aware of and utilizes all cluster resources for parallel rollback operations.

While the default behavior is to defer transaction recovery, you could choose to configure your system so transaction recovery completes before allowing client transactions to progress. In this scenario, the ability of Real Application Clusters to parallelize transaction recovery across multiple nodes is a more visible user benefit.

High Availability Configurations

This section discusses these Real Application Clusters high availability configurations:

- Default n-node Configuration
- Basic High Availability Configurations
- Shared High Availability Node Configuration

Default *n*-node Configuration

The Real Application Clusters *n*-node configuration is the default environment. All nodes of the cluster participate in client transaction processing and client sessions can be load balanced at connect time. Response time is optimized for available cluster resources, such as CPU and memory, by distributing the load across cluster nodes to create a highly available environment.

Benefits of *n*-Node Configurations

In the event of node failures, an instance on another node performs the necessary recovery actions. The database clients on the failed instance can be load balanced across the surviving (*n*-1) instances of the cluster. The increased load on each of the surviving instances can be minimized and availability increased by keeping response times within acceptable bounds. In this configuration, the database application workload can be distributed across all nodes and therefore provide optimal utilization of cluster machine resources.

Basic High Availability Configurations

You can easily configure a basic high availability system for Real Application Clusters in two-node environments. The primary instance on one node accepts user connections while the **secondary instance** on the other node accepts connections when the primary node fails, or when specifically selected through the `INSTANCE_ROLE parameter`. You can configure this manually by controlling the routing of transactions to specific instances. However, Real Application Clusters provides the **Primary/Secondary Configuration** feature to accomplish this automatically.

Configure the Primary/Secondary Instance feature by setting the **initsid.ora** parameter `ACTIVE_INSTANCE_COUNT` to `1`. In the two-node environment, the instance that first mounts the database assumes the **primary instance role**. The other instance assumes the role of **secondary instance**. If the **primary instance** fails, then the secondary instance assumes the primary role. When the failed instance returns to active status, it assumes the secondary instance role.

Remote Clients

The secondary instance becomes the primary instance only after the Cluster Manager informs it about the failure of the primary instance but before Global Cache Service and Global Enqueue Service reconfiguration and cache and transaction recovery processes begin. The redirection to the surviving instance happens transparently; application programming is not required. You only need to make minor configuration changes to the client connect strings.

In the Primary/Secondary Configuration, both instances run concurrently, like in any *n*-node Real Application Clusters environment. However, database application users only connect to the designated primary instance. The primary node masters all of the Global Cache Service and Global Enqueue Service resources. This minimizes communication between the nodes and provides performance levels that are nearly comparable to a traditional single node database.

The secondary instance can be used by specially configured clients, known as administrative clients, for batch query reporting operations or database administration tasks. This enables some level of utilization of the second node. It might also help off-load CPU capacity from the primary instance and justify the investment in redundant nodes.

The Primary/Secondary Configuration feature works in both dedicated server and shared server environments. However, it functions differently in each as described in the following sections:

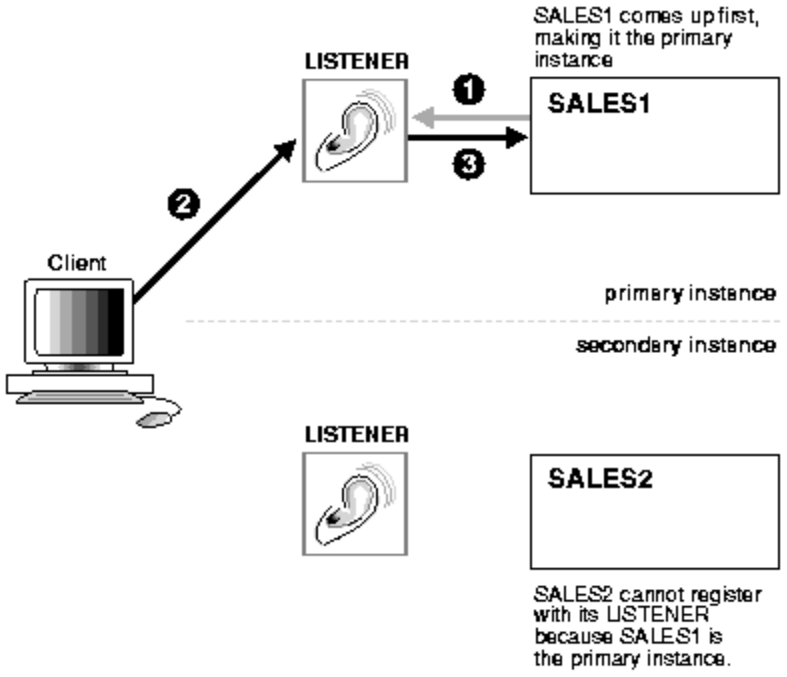
Primary/Secondary Instance in Dedicated Server Environments

In current high availability configurations, dedicated server environments do not use cross-instance listener registration. Connection requests made to a specific instance's listener can only be connected to that instance's service. This behavior is similar to the default *n*-node configuration in dedicated server environments.

Figure 10-3 shows a cluster configuration before a node failure.

1. SALES1 is in contact with a listener.
2. A client is in contact with a listener.
3. SALES1 becomes the primary instance.
4. Soon after, SALES2 will become the secondary instance.

Figure 10-3 Primary/Secondary Instance Feature in Dedicated Server Environments

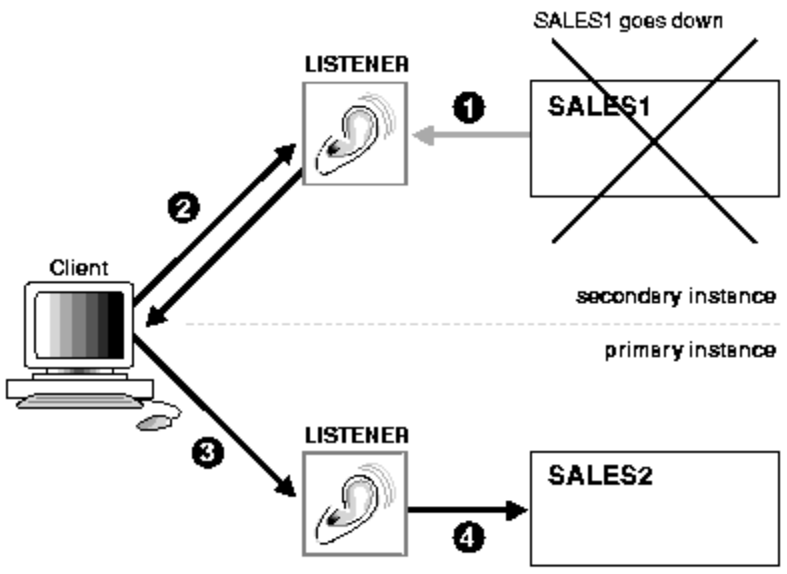

SALES1 comes up first, making it the primary instance
LISTENER
1
SALES1
3
2
Client
primary instance
secondary instance
LISTENER
SALES2
SALES2 cannot register with its LISTENER because SALES1 is the primary instance.

Text description of the illustration pss81016.gif

When the primary instance fails, as shown in Figure 10-4, the following steps occurs:

1. The failure of SALES1 is noted.
2. A reconnection request from the client is rejected by the failed instance's listener.
3. The secondary instance performs recovery and becomes the primary instance.
4. Upon resubmitting the client request, the client reestablishes the connection by using the new primary instance's listener that then connects the client to the new primary instance. Note that the connection is reestablished automatically when you use address lists or if your client is configured to use connection failover.

Figure 10-4 Node Failure in Dedicated Server Environments



Text description of the illustration pss81017.gif

Primary/Secondary Instance Feature and the Shared Server

Real Application Clusters provides reconnection performance benefits when running in shared server mode. This is accomplished by the cross-registration of all the dispatchers and listeners in the cluster.

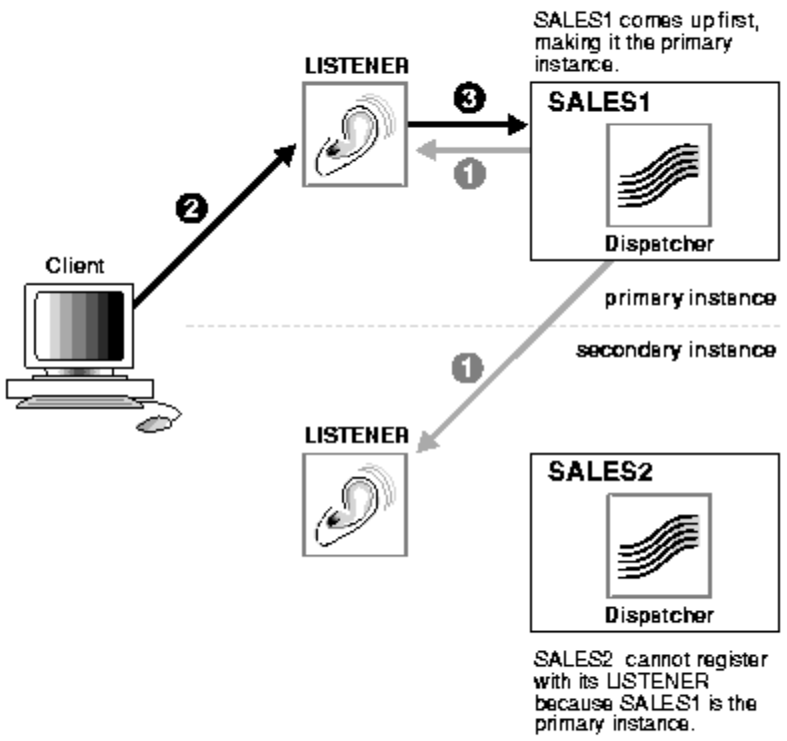
In the Primary/Secondary configurations, the primary instance's dispatcher registers as the primary instance with both listeners, as shown in Figure 10-5:

- A client could connect to either listener. (Only the connection to the primary node listener is illustrated).
- The client contacts the listener.
- The listener then connects the client to the dispatcher. (Only the listener/dispatcher connection on the primary node is illustrated).

See Also:

Oracle9i Real Application Clusters Installation and Configuration for information on configuring client connect strings

Figure 10-5 Primary/Secondary Instance Feature in Shared Server Environments


SALES1 comes up first, making it the primary instance.
LISTENER
3
SALES1
1
2
Dispatcher
Client
primary instance
secondary instance
1
LISTENER
SALES2
Dispatcher
SALES2 cannot register with its LISTENER because SALES1 is the primary instance.

Text description of the illustration pss81019.gif

Specially configured clients can use the secondary instance for batch operations. For example, batch reporting tasks or index creation operations can be performed on the secondary instance.

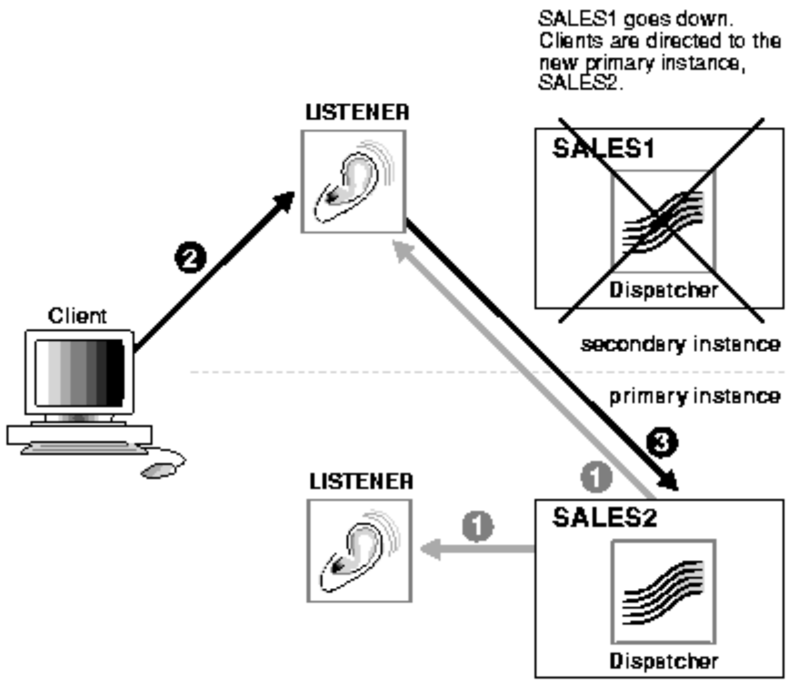
See Also:

Oracle9i Real Application Clusters Installation and Configuration for instructions on how to connect to secondary instances

Figure 10-6 shows how a failed primary instance is replaced by a *new* primary instance.

1. If the primary node fails, then the dispatcher in the secondary instance registers as the *new* primary instance with the listeners.
2. The client requests a reconnection to the database through either listener.
3. The listener directs the request to the new primary instance's dispatcher.

Figure 10-6 Node Failure in Shared Server Environments



Text description of the illustration pss81018.gif

Warming the Library Cache on the Secondary Instance

Maintaining information about frequently executed SQL and PL/SQL statements in the library cache improves the performance of the Oracle database server. In Real Application Clusters primary and secondary instance configurations, the library cache associated with the primary instance contains up-to-date information. If failover occurs, then the benefit of that information is lost unless the library cache on the secondary instance is populated beforehand.

Use the `DBMS_LIBCACHE` package to transfer information in the library cache of the primary instance to the library cache of the secondary instance. This process is called **warming the library cache**. It improves performance immediately after failover because the new primary library cache does not need to be populated with parsed SQL statements and compiled PL/SQL units.

See Also:

Oracle Real Application Clusters Guard Administration and Reference Guide for more information about installing and configuring the warming the library cache feature and *Oracle9i Supplied PL/SQL Packages Reference* for more information about using `DBMS_LIBCACHE`

Benefits of Basic High Availability Configurations

The Primary/Secondary Instance feature provides two important reasons for using this scenario instead of a default two-node configuration. The Primary/Secondary Instance feature provides:

- A viable transition path for upgrading to an *n*-node configuration
- A highly available solution for applications that do not need to scale beyond one node
- Performance that is comparable to single instance databases
- High performance without requiring tuning

Transition Path to *n*-node Configurations

The Primary/Secondary Instance feature offers a gradual way to migrate from a single instance application environment to a Real Application Clusters environment. This configuration also minimizes tuning issues because all client transactions are performed on one node at any given time. It also simplifies troubleshooting system problems because you tune only one node at a time as opposed to simultaneously tuning two or more nodes.

Since availability is also dependent on the DBA's ability to manage, tune, and troubleshoot the environment, the Real Application Clusters Primary/Secondary Instance feature provides a gradual way to ease the database administration staff into using Real Application Clusters.

Shared High Availability Node Configuration

Running Real Application Clusters in an *n*-node configuration optimally utilizes the cluster resources. However, as discussed previously, this is not always possible or advisable. On the other hand, the financial investment required to have an idle node for failover is often prohibitive. These situations might instead be best suited for a shared high availability node configuration.

This type of configuration typically has several nodes each running a separate application module or service where all application services share one Real Application Clusters database. You can configure a separate designated node as a failover node. While an instance is running on that node, no users are being directed to it during normal operation. In the event that any one of the application nodes fails, Oracle can direct the workload to the failover node.

While this configuration is a useful one to consider for applications that need to run on separate nodes, it works best if a middle tier application or transaction processing monitor directs the appropriate application users to the appropriate nodes. Unlike the Primary/Secondary Instance Configuration, there is no database setup that automates the workload transition to the failover node. The application, or middle-tier software, would need to direct users from the failed application node to the designated failover node. The application would also need to control failing back the users once the failed node is operational. Failing back frees the failover node for processing user work from subsequent node failures.

Benefits of Shared High Availability Node Configurations

In this configuration, application performance is maintained in the event of a failover. In the *n*-node configuration, application performance could degrade by 1/*n* due to the same workload being redistributed over a smaller set of cluster nodes.

Toward Deploying High Availability

Real Application Clusters on clustered systems provides a fully redundant environment that is extremely fault resilient. Architecture is central to high availability in Real Application Clusters environments. All cluster nodes have an active instance that has equal access to all the data. If a node fails, then all users have access to all data by way of surviving instances on the other nodes. In-progress transactions on the failed node are recovered by the first node that detects the failure. In this way, there is minimal interruption to end user application availability with Real Application Clusters.

Previous Next

Home Book List Contents Index Master Index Feedback