Oracle9*i* Real Application Clusters

Concepts

Release 1 (9.0.1)

June 2001

Part No. A89867-01

Oracle9*i* Real Application Clusters Concepts, Release 1 (9.0.1)

Part No. A89867-01

Primary Author: James W. Rawles.

Primary Contributors: David Austin, Mark Bauer, Mitch Flatland, Merrill Holt, and Michael Zoll.

Contributors: Lance Ashdown, Bill Bridge, Wilson Chan, Sashikanth Chandraserkaran, Sandra Cheever, Carol Colrain, Jonathan Creighton, Sohan DeMel, Connie Dialeris, Xing Du, Mitch Flatland, Deepak Gupta, Mike Hartstein, Eugene Ho, Wei Hu, Ashok Joshi, Bill Kehoe, Susan Kim, Jonathan Klein, Raj Kumar, Tirthankar Lahiri, Bill Lee, Sajjad Masud, Neil Macnaughton, Rita Moran, Kotaro Ono, Kant Patel, Stefan Pommerenk, Mark Porter, Darryl Presley, Brian Quigley, Rebecca Reitmeyer, Ann Rhee, Joao Rimoli, Pat Ritto, Roger Sanders, Daniel Semler, Vinay Srihari, Alok Srivastava, Bob Thome, Alex Tsukerman, Tak Wang, Shari Yamaguchi, and Tolga Yurek.

Graphic Designer: Valarie Moore.

Contents

List of Figures

List of Tables

Send Us Your Comments

Oracle9*i* Real Application Clusters Concepts, Release 1 (9.0.1)

Part No. A89867-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title and part number, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: infodev_us@oracle.com
- FAX: (650) 506-7227   Attn: Server Technologies Documentation Manager
- Postal service:
  Oracle Corporation
  Server Technologies Documentation
  500 Oracle Parkway, Mailstop 4op11
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual prepares you to successfully implement clustered databases by presenting Real Application Clusters concepts. Information in this manual applies to Real Application Clusters as it runs on all operating systems.

> **Note:** Beginning with Release 1 (9.0.1), Oracle Parallel Server was renamed Real Application Clusters

You should read this manual before reading *Oracle9i Real Application Clusters Installation and Configuration*, *Oracle9i Real Application Clusters Administration*, and *Oracle9i Real Application Clusters Deployment and Performance*. For general information about Oracle and administering the single **instance** Oracle Server, refer to *Oracle9i Database Concepts* and *Oracle9i Database Administrator's Guide.*

This preface contains these topics:

- Audience
- Organization
- Related Documentation
- Conventions

Audience

Real Application Clusters Concepts is intended for database administrators and application developers who work with Real Application Clusters.

Organization

This book presents Real Application Clusters concepts in five parts. It begins by describing cluster database processing fundamentals for Real Application Clusters. The book then covers resource processing among instances. Next, it explains the fundamentals of how Real Application Clusters is implemented. Next, it describes High Availability and the Oracle Real Application Clusters Guard feature. It ends with reference material that includes an appendix describing the implementation restrictions on Real Application Clusters, an appendix on pre-Release 1 (9.0.1) lock setting concepts, and a glossary of terms.

Part I "Cluster Database Processing Fundamentals"

Chapter 1, "Introduction to Real Application Clusters"

This chapter describes prerequisite knowledge, discusses terminology, and introduces cluster database processing and database technologies that offer advantages for online transaction processing and decision support applications.

Chapter 2, "Real Application Clusters Architecture"

This chapter describes the architectural components that Oracle provides for cluster database processing.

Chapter 3, "Cluster Hardware Architecture"

This chapter describes the hardware components and high-level architectural models that typify cluster environments.

Part II "Resource Coordination in Real Application Clusters"

Chapter 4, "Local Resource Coordination"

This chapter describes inter-instance coordination activities involved in a cluster and local resources employed by the Real Application Clusters software.

Chapter 5, "Cache Fusion and the Global Cache Service"

This chapter provides a detailed discussion of Cache Fusion and the inter-instance coordination activities handled by the **Global Cache Service (GCS)**.

Chapter 6, "Coordination by the Global Enqueue Service"

This chapter provides details on and the inter-instance coordination activities handled by the **Global Enqueue Service (GES)**.

Part III "Implementing Real Application Clusters"

Chapter 7, "Real Application Clusters Components"

This chapter describes the implementation components for Real Application Clusters applications.

Chapter 8, "Real Application Clusters Storage Considerations"

This chapter describes the storage considerations for Real Application Clusters applications.

Chapter 9, "Scalability in Real Application Clusters"

This chapter describes the **scalability** features of Real Application Clusters.

Part IV "High Availability"

Chapter 10, "High Availability Concepts and Best Practices"

This chapter describes the concepts and some best practices methodologies for how to use Real Application Clusters to implement high availability.

Chapter 11, "Oracle Real Application Clusters Guard Architecture"

This chapter describes the architecture of Oracle Real Application Clusters Guard.

Chapter 12, "Oracle Real Application Clusters Guard Operation"

This chapter describes the operation of Oracle Real Application Clusters Guard.

Part V "Reference"

Appendix A, "Restrictions"

This appendix lists restrictions of Real Application Clusters software.

Appendix B, "Using Pre-Release 1 (9.0.1) Multi-Block Lock Assignments (Optional)"

This appendix describes how to use the optional pre-Release 1 (9.0.1) lock setting to override the concurrency control provided by Cache Fusion.

Glossary

A glossary of Real Application Clusters and Oracle Real Application Clusters Guard terms.

Related Documentation

After reading this manual, read *Oracle9i Real Application Clusters Installation and Configuration*, *Oracle9i Real Application Clusters Deployment and Performance*, and *Oracle9i Real Application Clusters Administration*. You can also read the *Oracle Real Application Clusters Guard Administration and Reference Guide* for additional information on Oracle Real Application Clusters Guard.

> **See Also:** The *Oracle9i Real Application Clusters Documentation Online Roadmap* can help you use the online Real Application Clusters Documentation set

For more information, see these Oracle resources:

Installation Guides

- *Oracle9i Installation Guide* for Compaq Tru64, Hewlett-Packard HPUX, IBM AIX, Linux, and Sun Solaris-based systems
- *Oracle9i Database installation guide for Windows*
- *Oracle Diagnostics Pack Installation*
- Oracle9*i* Real Application Clusters Guard installation manuals (several titles available, platform specific)

Operating System-Specific Administrator's Guide

- *Oracle9i Database Administrator's Guide* for Compaq Tru64, Hewlett-Packard HPUX, IBM AIX, Linux, and Sun Solaris-based systems

Cluster Database Management

- *Oracle Enterprise Manager Administrator's Guide*
- *Getting Started with the Oracle Diagnostics Pack*

Oracle Server Documentation

- *Oracle9i Database New Features*
- *Oracle9i Database Concepts*
- *Oracle9i Database Administrator's Guide*
- *Oracle9i Database Reference*
- *Oracle Net Services Administrator's Guide*

In North America, printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

Customers in Europe, the Middle East, and Africa (EMEA) can purchase documentation from

`http://www.oraclebookshop.com/`

Other customers can contact their Oracle representative to purchase printed documentation.

To download free release notes, installation documentation, white papers, or other collateral material, please visit the Oracle Technology Network (OTN) Web site. You must register online before using OTN. Registration is free, and can be done at

`http://technet.oracle.com/membership/index.htm`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://technet.oracle.com/docs/index.htm`

Conventions

This section describes the conventions used in the text and code examples of the this documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	The C datatypes such as **ub4**, **sword**, or **OCINumber** are valid. When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles, emphasis, syntax clauses, or placeholders.	*Oracle9i Database Concepts* You can specify the *parallel_clause*. Run `U`*`old_release`*`.SQL` where *old_release* refers to the release you installed prior to upgrading.
UPPERCASE monospace (fixed-width font)	Uppercase monospace typeface indicates elements supplied by the system. Such elements include parameters, privileges, datatypes, RMAN keywords, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Specify the `ROLLBACK_SEGMENTS` parameter. Use the `DBMS_STATS.GENERATE_STATS` procedure.
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables and sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values.	Enter `sqlplus` to open SQL*Plus. The `department_id`, `department_name`, and `location_id` columns are in the `hr.departments` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true.` Connect as `oe` user.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a monospace (fixed-width) font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`

|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}`
		`[COMPRESS | NOCOMPRESS]`

...	Horizontal ellipsis points indicate either:	
	■ That we have omitted parts of the code that are not directly related to the example	`CREATE TABLE ... AS subquery;`
	■ That you can repeat a portion of the code	`SELECT col1, col2, ... , col`*n* `FROM employees;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	

Other notation	You must enter symbols other than brackets, braces, vertical bars, and ellipsis points as it is shown.	`acctbal NUMBER(11,2);`
		`acct    CONSTANT NUMBER(4) := 3;`

Italics	Italicized text indicates variables for which you must supply particular values.	`CONNECT SYSTEM/`*system_password*

Convention	Meaning	Example

`UPPERCASE`	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT last_name, employee_id FROM employees;`
		`SELECT * FROM USER_TABLES;`
		`DROP TABLE hr.employees;`

`lowercase`	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.	`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`

Documentation Accessibility

Oracle's goal is to make our products, services, and supporting documentation accessible to the disabled community with good usability. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at

`http://www.oracle.com/accessibility/`

JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

What’s New in Real Application Clusters?

This section describes how the features, variables, parameters, views, and installation procedures for Real Application Clusters have changed in recent releases.

Release 1 (9.0.1) New Features in Real Application Clusters

Beginning with Release 1 (9.0.1), Oracle Parallel Server was renamed Oracle Real Application Clusters.

This release introduces a new phase of **Cache Fusion**, a breakthrough technology that guarantees **cache coherency** among multiple **cluster** nodes without incurring disk **I/O** costs. The first phase of Cache Fusion was introduced in Oracle8*i* to improve read / write concurrent data access. This release extends that capability to optimize read / read, read / write, and write / write concurrency among multiple cluster **node**s. The new features greatly enhance Real Application Clusters performance and **scalability**.

Real Application Clusters introduces a number of significant improvements. These improvements are divided into several categories:

- New Terminology
- Installation and Configuration
- Cache Fusion and Resource Management
- SRVCTL Utility
- Listener Load Balancing and Failover for Dedicated Servers
- Diagnostic Features

- Discovery
- High Availability Features
- Shutdown Transactional Local Command
- Quiesce Database Feature
- New SQL Script Names
- TRACE_ENABLED Parameter
- Changed Parameter Names
- Obsolete Parameters
- New View Names
- Revised View Names

New Terminology

Starting with Release 1 (9.0.1), a number of terms have been revised to more accurately reflect new functionality:

- Oracle Parallel Server is now called **Real Application Clusters**.
- Parallel server databases are now called **cluster databases**.
- Oracle Parallel Fail Safe (OPFS) was renamed **Oracle Real Application Clusters Guard**.
- The term Distributed Lock Manager (DLM) is obsolete. The background processes that were formerly aggregated under the DLM still exist, but the DLM no longer manages resources in Real Application Clusters. The **Global Cache Service (GCS)** and **Global Enqueue Service (GES)** now handle the management functions that had been handled by the DLM in earlier releases.

  Note: The terms for background processes (LMON, LMD, LMSn, and so on) remain unchanged to ensure backward compatibility.

- The OPSCTL utility was renamed **SRVCTL**.
- The Oracle Parallel Server Communications Daemon (OPSD) was renamed the **Global Services Daemon (GSD)**.

- Oracle Parallel Server Management (OPSM) was renamed **Server Management (SRVM)**
- The term Parallel Cache Management (PCM) is obsolete. These operations have been replaced by Global Cache Service block requests.
- The term non-PCM is obsolete. The Global Enqueue Service now implements the resource management functions formerly called non-PCM.
- Lock mastering is obsolete. The equivalent function in Release 1 (9.0.1) is **resource mastering**.
- The Lock Database is obsolete. The equivalent functions are handled by the **Global Resource Directory**.

Installation and Configuration

- There are new raw device requirements in Release 1 (9.0.1). You must create a shared raw device to store Real Application Clusters server configuration information. In previous releases, Oracle stored this information in the `db_name.conf` file on UNIX and in the Registry on Windows NT platforms.
- If you use the preconfigured database templates, you must also create several shared raw devices that were optional in previous releases. Create these raw devices for the Oracle9*i* Sample Schemas, Online Analytical Processing (OLAP), Oracle9*i* interMedia, and for the automatic undo management feature. These requirements are in addition to the raw device requirements of previous releases, such as those for the `SYSTEM`, `USERS`, `TEMP`, and other tablespaces.
- The Oracle Universal Installer and the Oracle **Database Configuration Assistant (DBCA)** have been enhanced to offer simplified software installation and database creation for Real Application Clusters.
  - The Oracle Universal Installer offers the following database configuration types: General Purpose, Transaction Processing, Data Warehouse, Customized, and Software Only. The Customized database configuration type replaces the Custom installation type.
  - The DBCA also has templates named General Purpose, Transaction Processing, and Data Warehousing. These templates correspond to the Installer's configuration types. In addition, the DBCA has a New Database template that does not include datafiles and that is fully customizable.
  - To simplify installation, you can select the General Purpose, Transaction Processing, or Data Warehousing configuration type in the Universal Installer's Database Configuration screen. Providing you have configured

your shared raw devices as described in *Oracle9i Real Application Clusters Installation and Configuration*, after completing the remaining Installer screens, the installation and DBCA database creation processes continue without further user input.

> **See Also:** *Oracle9i Real Application Clusters Installation and Configuration* for details on changes to configuration types, raw partitions, templates, Oracle Universal Installer, and DBCA options.

- The **raw partition** tablespace size requirements changed for Release 1 (9.0.1). These tablespaces require greater capacities than the values that were published in the release 8.1.7 documentation.

> **See Also:** *Oracle9i Real Application Clusters Installation and Configuration* and the Release 1 (9.0.1) README file for details on raw partition tablespace size requirements

- Release 1 (9.0.1) supports cluster file systems to simplify Real Application Clusters installation and management.

> **See Also:** *Oracle9i Real Application Clusters Administration* for information on cluster file systems

- If you use the Oracle Managed Files option, your hardware server platform must support a cluster file system, and this cluster file system must be supported by Oracle. The Oracle Managed Files feature automatically creates and deletes files that Oracle requires to manage the database. Cluster file systems are only available for a limited number of system types.

> **See Also:**
>
> - *Oracle9i Real Application Clusters Administration* and *Oracle9i Database Administrator's Guide* for information on Oracle Managed Files
> - *Oracle9i Real Application Clusters Installation and Configuration*

- The Oracle Cluster Setup Wizard is available to install the **Operating System-Dependent (OSD) Layer** software on Windows NT and Windows 2000 operating systems.

See Also: *Oracle9i Database Installation Guide for Windows* for information on OSD software installation

---

Note: The term Windows is used generically in this manual for Microsoft Windows products. Real Application Clusters is compatible with the Windows NT and Windows 2000 operating systems, but not with Windows 95, Windows 98, or Windows ME.

- Real Application Clusters and Oracle Fail Safe are integrated with the Microsoft's Cluster Server (MSCS). This enables more flexibility in configuring them on Windows NT. They also make use of Internet Protocol (IP) failover and Microsoft's cluster management utilities.

  See Also:

  - Oracle Fail Safe documentation for configuration instructions
  - Microsoft Corporation documentation at www.microsoft.com for information on Microsoft's Cluster Server.

- The Oracle Database Configuration Assistant (**DBCA**) has administrative features such as **instance** and template management. Instance management enables you to add or delete instances. You can use the template management feature to manage and customize database creation scripts.

  See Also: *Oracle9i Real Application Clusters Installation and Configuration* for information on the DBCA

- The DBCA provides tools to create databases. As part of creating databases, it can create and display the contents server side server parameter files (SPFILEs).

  Oracle Corporation recommends that you use server parameter files when you implement Real Application Clusters.

  Oracle Corporation also recommends that you modify the default file location for server parameter files for Real Application Clusters environments if you are using raw devices.

See Also:

- Chapter 8, "Real Application Clusters Storage Considerations"
- *Oracle9i Real Application Clusters Installation and Configuration*

- Beginning with Release 1 (9.0.1), default user names expire. All default user names *except* `SYS`, `SYSTEM`, and `SCOTT`, expire upon installation. To use these names, you must explicitly unlock them.

Cache Fusion and Resource Management

- Cache Fusion is fully implemented in Real Application Clusters. This phase eliminates **forced disk write**s. Fixed locks have been eliminated. Although not normally recommended, it is possible to override the default configuration and implement pre-9.0.1 releasable locks in special circumstances. However, the default Cache Fusion concurrency control provides exceptional performance and scalability for most applications.

  See Also:

  - Chapter 5, "Cache Fusion and the Global Cache Service" for a description of Cache Fusion processing
  - Appendix B, "Using Pre-Release 1 (9.0.1) Multi-Block Lock Assignments (Optional)" for information on using pre-9.0.1 locks

- The new **dynamic resource remastering** feature is implemented in Real Application Clusters. Dynamic resource remastering is the ability of the Global Cache Service to move the ownership of a resource between instances of Real Application Clusters during runtime without affecting availability.

  See Also: Chapter 2, "Real Application Clusters Architecture" for a description of Dynamic resource remastering

Real Application Clusters now employs **resource affinity**. Resource affinity is the use of dynamic resource remastering to move the location of the resource masters for a database file to the instance where operations are most frequently occurring.

See Also: Chapter 2, "Real Application Clusters Architecture" for a description of resource affinity

- Multiple **Global Cache Service Processes (LMSn)** are available to increase scalability for Global Cache Service processing. The number of LMS processes varies depending on the load and system resources. The **Global Enqueue Service Daemon (LMD)** process has been superseded by LMS*n* processes under most circumstances.

  See Also: Chapter 5, "Cache Fusion and the Global Cache Service" for information on LMS*n* processes.

- Block Server Processes (BSP*n*) were eliminated. Their former functionality is now handled by **Global Cache Service Processes (LMSn)**.

  See Also: Chapter 5, "Cache Fusion and the Global Cache Service" for information on LMSn processes.

- The **LCK process** has been simplified. Much of the obsolete LCK functionality is now handled by **Global Cache Service Processes (LMSn)**. In Real Application Clusters, there is just one remaining LCK process. It is used for non-Cache Fusion operations such as instance requests and cross-instance call operations.

  See Also: Chapter 5, "Cache Fusion and the Global Cache Service" for information on LMSn processes.

SRVCTL Utility

- The OPSCTL utility was renamed **SRVCTL** and is fully functional in Release 1 (9.0.1) Real Application Clusters. Oracle Corporation recommends that you use SRVCTL to administer your Real Application Clusters database environment. SRVCTL manages configuration information that is used by several Oracle tools. For example, Oracle Enterprise Manager and the Oracle Intelligent Agent use the configuration information that SRVCTL generates to discover and monitor nodes in your cluster.

  See Also: *Oracle9i Real Application Clusters Administration* for information on SRVCTL

- The additional *instance*_startup (**Transport Network Services (TNS)** entries that had been required for each instance are now obsolete. (These had been required in release 8.1 for OPSCTL to start remote instances).

Storage

- The automatic undo management mode offers the full range of functionality in Real Application Clusters as it does with single instance Oracle. That is, it manages undo **tablespace**s. With ALTER_SYSTEM, you can dynamically redirect an instance's undo processing from one undo tablespace to another.

  See Also: *Oracle9i Real Application Clusters Administration* for information on the Automatic Undo Management mode

Oracle Enterprise Manager

- **Oracle Enterprise Manager** has improved abilities to discover, start up, and shut down Real Application Clusters.

  See Also: *Oracle9i Real Application Clusters Administration* for information on Oracle Enterprise Manager

Listener Load Balancing and Failover for Dedicated Servers

- The Transport Network Services (TNS) listener previously provided load balancing across nodes only in **shared server**s (formerly called multithreaded servers). Starting with Release 1 (9.0.1), load balancing and failover is provided for dedicated server configurations.

  See Also: *Oracle Net Services Administrator's Guide* for details on load balancing

Diagnostic Features

- A SID entry in the server parameter file identifies parameter settings that are different on specific instances within Real Application Clusters environments.

  See Also: Chapter 8, "Real Application Clusters Storage Considerations" and *Oracle9i Real Application Clusters Installation and Configuration* for information on server parameter files

- Release 1 (9.0.1) provides an improved Oracle Enterprise Manager console for Real Application Clusters. Oracle Enterprise Manager's Performance Manager and Capacity Planner have 23 new Oracle Enterprise Manager statistics charts that are specific to Real Application Clusters.

> **See Also:** *Oracle9i Real Application Clusters Deployment and Performance* for details on the statistics charts

- Real Application Clusters now employs a **diagnosability daemon**. This is a background process that captures diagnostic data on instance process failures. No user control is required for this daemon.

  > **See Also:** Chapter 2, "Real Application Clusters Architecture" for a description of the diagnosability daemon

Discovery

- Database discovery has been integrated into the Oracle Intelligent Agent. The **Database Configuration Assistant (DBCA)** can add, remove, or rename instances from cluster databases.

  > **See Also:** *Oracle9i Real Application Clusters Administration* for information on the Oracle Intelligent Agent and the DBCA.

- Release 1 (9.0.1) has made significant revisions to Auto Discovery, including Java implementations of the **Global Services Daemon (GSD)** (formerly called OPSD) for Unix and NT. Note that GSD is installed by default.

  > **See Also:** Chapter 2, "Real Application Clusters Architecture" for information on GSD

High Availability Features

Beginning with Release 1 (9.0.1), Oracle Parallel Fail Safe (OPFS) was renamed **Oracle Real Application Clusters Guard**. It became a feature of Real Application Clusters on UNIX platforms. Oracle Real Application Clusters Guard has these advantages:

- A simplified **pack** structure
- Takes advantage of Real Application Clusters instance roles. This enables preservation of primary/secondary roles. The **resource mastering** function is done locally on the primary instance. This improving performance.
- Packs (including IP addresses) float, minimizing TCP/IP timeouts
- Improved performance for pack **failover**

- Improved error and message logging
- Notifications are logged and sent when instances start up, shut down, or change roles
- Globalization Technology (formerly called National Language Support or NLS)
- Uptime and downtime logging and reporting capability
- Improved user interface and commands
- Improved installation

  See Also: Chapter 11 and Chapter 12 for information on Oracle Real Application Clusters Guard

- Release 1 (9.0.1) reduces the time required to remap resources following failover.

  See Also: Chapter 10, "High Availability Concepts and Best Practices" for information on high availability and failover recovery

- Release 1 (9.0.1) has a new **warming the library cache** feature

To improve failover performance in Primary / Secondary instance configurations, use the `DBMS_LIBCACHE` package to transfer information from the library cache of the primary instance to the library cache of the secondary instance.

See Also:

- Chapter 10, "High Availability Concepts and Best Practices" and *Oracle Real Application Clusters Guard Administration and Reference Guide* for more information about the warming the library cache feature
- *Oracle9i Supplied PL/SQL Packages Reference* for more information about using `DBMS_LIBCACHE`

- **Recovery Manager (RMAN)** has new features that improve the manageability of backups and archive logs, and allow persistent configuration for many commonly used RMAN options.

  See Also: *Oracle9i Recovery Manager User's Guide and Reference* for information on RMAN

- The Block Media Recovery (BMR) feature enables you to recover specific data blocks.

  See Also: *Oracle9i Recovery Manager User's Guide and Reference* for information on BMR

Shutdown Transactional Local Command

- The `SHUTDOWN TRANSACTIONAL LOCAL` command was introduced as an option to the existing `SHUTDOWN TRANSACTIONAL` command. This new command can be used to prevent new transactions from starting locally, and to perform an immediate shutdown after all local transactions have completed. With the new command, you can gracefully move all sessions from one instance to another by shutting down selected instances transactionally.

Quiesce Database Feature

- The `QUIESCE RESTRICTED` clause was added. This new feature enables you to perform administrative tasks in isolation from concurrent user transactions or queries.

  See Also: Chapter 10, "High Availability Concepts and Best Practices" and *Oracle9i Real Application Clusters Administration* for information on the Quiesce Database feature

New SQL Script Names

A number of SQL script names were changed to correspond to the change in product name from Oracle Parallel Server to Real Application Clusters.

- The `utlopslt.sql` script name was changed to `utlclust.sql`
- The `catparr.sql` script name was changed to `catclust.sql`
- The `ops.sql` script name was changed to `clustdb.sql`

TRACE_ENABLED Parameter

The `TRACE_ENABLED` parameter was added in Release 1 (9.0.1). When enabled, this dynamic parameter provides low overhead memory tracing. It is enabled by default.

See Also: *Oracle9i Real Application Clusters Deployment and Performance* for information on the `TRACE_ENABLED` parameter.

Changed Parameter Names

A number of parameter names were changed to correspond to the change in product name from Oracle Parallel Server to Real Application Clusters.

- The OPS_INTERCONNECTS parameter name was changed to CLUSTER_INTERCONNECTS. Note that with Sun Clusters configurations the interconnect High Availability feature is not available.
- The PARALLEL_SERVER parameter name was changed to CLUSTER_DATABASE.
- The PARALLEL_SERVER_INSTANCES parameter name was changed to CLUSTER_DATABASE_INSTANCES.

Note: If you use any of the old parameter names, you will likely get a warning message, telling you that the parameter name has been deprecated. If you have implemented the new parameter name, you can ignore these warning messages.

The CLUSTER_DATABASE parameter is required. The CLUSTER_DATABASE_INSTANCES parameter, however, is optional. It can be used for optimization

See Also: *Oracle9i Real Application Clusters Deployment and Performance* for additional information on how to use the new CLUSTER_INTERCONNECTS,CLUSTER_DATABASE, and CLUSTER_DATABASE_INSTANCES parameters.

Obsolete Parameters

The following parameters are obsolete in Release 1 (9.0.1):

- GC_DEFER_TIME
- GC_RELEASABLE_LOCKS
- GC_ROLLBACK_LOCKS
- LM_LOCKS
- LM_RESS

New View Names

The following views have been added in Release 1 (9.0.1). These views track the current and previous master instances and the number of re-masterings of enqueue (`V$HVMASTER_INFO`), global cache (`V$GCSHVMASTER_INFO`), and global cache resources belonging to a file accessed frequently by a single instance (`V$GCSPFMASTER_INFO`):

- `V$HVMASTER_INFO` for Global Enqueue Service resources
- `V$GCSHVMASTER_INFO` for Global Cache Service resources except those belonging to files mapped to a particular master.
- `V$GCSPFMASTER_INFO` for Global Cache Service resources belonging to files mapped to a particular master.

Revised View Names

The following views have been re-named in Release 1 (9.0.1):

`V/GV$DLM_MISC` is now `V/GV$GES_STATISTICS`

`V/GV$DLM_LATCH` is now `V/GV$GES_LATCH`

`V/GV$DLM_CONVERT_LOCAL` is now `V/GV$GES_CONVERT_LOCAL`

`V/GV$DLM_CONVERT_REMOTE` is now `V/GV$GES_CONVERT_REMOTE`

`V/GV$DLM_ALL_LOCKS` is now `V/GV$GES_ENQUEUE`

`V/GV$DLM_LOCKS` is now `V/GV$GES_BLOCKING_ENQUEUE`

`V/GV$DLM_RESS` is now `V/GV$GES_RESOURCE`

`V/GV$DLM_TRAFFIC_CONTROLLER` is now `V/GV$GES_TRAFFIC_CONTROLLER`

`V/GV$LOCK_ELEMENT` is now `V/GV$GC_ELEMENT`

`V/GV$BSP` is now `V/GV$CR_BLOCK_SERVER`

`V/GV$LOCKS_WITH_COLLISIONS` is now `V/GV$GC_ELEMENTS_WITH_COLLISIONS`

`V/GV$FILE_PING` is now `V/GV$FILE_CACHE_TRANSFER`

`V/GV$TEMP_PING` is now `V/GV$TEMP_CACHE_TRANSFER`

`V/GV$CLASS_PING is now V/GV$CLASS_CACHE_TRANSFER`

`V/GV$PING` is now `V/GV$CACHE_TRANSFER`

Oracle8*i* Release 3 (8.1.7) New Features

Beginning with release 8.1.7, Oracle Parallel Server had the following changes:

- Raw Partition Tablespace Size Requirements

Some of the **raw partition** tablespace size requirements changed for Oracle Parallel Server release 8.1.7 as shown in the following table. These tablespaces require slightly greater capacities than the values that were published in release 8.1.6 documentation.

Create a Raw Device for	With File Size
SYSTEM	275 MB
TEMP	78 MB for Online Transaction Processing environments 520 MB for Decision Support Systems
DRSYS	96 MB

- Connecting to Secondary Instances

The process of configuring Oracle Parallel Server to connect to secondary instances was simplified for release 8.1.7. Use the `INSTANCE_ROLE` parameter in the Connect Data portion of the connect descriptor to configure explicit secondary instance connections.

Recovery Manager Procedures for Oracle Parallel Server

The procedures for connecting Recovery Manager (RMAN) to a target database in an Oracle Parallel Server cluster changed for release 8.1.7. For more information refer to *Oracle9i Recovery Manager Reference*.

- OPS_INTERCONNECTS Parameter for Solaris

`OPS_INTERCONNECTS` provides information about additional cluster interconnects for use in Oracle Parallel Server environments. Oracle uses the information from this parameter to distribute traffic among the various interfaces. You would normally use `OPS_INTERCONNECTS` when a single interconnect is insufficient to meet the bandwidth requirements of large Oracle Parallel Server databases.

`OPS_INTERCONNECTS` is an optional parameter. If you do not set it, then the current semantics that determine the appropriate interconnect for Oracle Parallel Server internode communication are preserved.

The syntax of the parameter is:

```
OPS_INTERCONNECTS = if1:if2:...:ifn
```

Where *`ifn`* is an IP address in standard dotted-decimal format, for example, `144.25.16.214`. Subsequent platform implementations may specify interconnects with different syntaxes.

Note that when you set `OPS_INTERCONNECTS` in Sun Cluster configurations, the interconnect High Availability features are not available. In other words, an interconnect failure that is normally unnoticeable would instead cause an Oracle cluster failure.

Oracle8*i* Release 2 (8.1.6) New Features

Beginning with release 8.1.6, Oracle Parallel Server had the following changes:

- Primary / Secondary Feature

With the **Primary/Secondary Configuration** feature you can implement a basic high availability configuration using the Primary / Secondary Instance feature. This feature serves two-node Oracle Parallel Server environments. The primary instance on one node accepts user connections while the secondary instance on the other node only accepts connections when the primary node fails.

- Additional Statistics

The following statistics were added:

  - global cache cr block send time – total time to send the block.
  - global cache cr block log flushes – number of log flushes.
  - global cache cr block log flush time – total time for log flushes.
  - global cache prepare failures – the number of prepares that failed.

- Obsolete Statistics

The following statistics became obsolete:

  - global cache consistent read from disk
  - global cache fairness down converts

- Changes in Default Parameter Settings

The default setting for `GC_ROLLBACK_LOCKS` is "`0-128=32!8REACH`" This protects rollback segments 0 through 129 with locks.

- LM_LOCKS and LM_RESS Automatically Set by Oracle

Oracle automatically sets values for `LM_LOCKS` and `LM_RESS` based on settings in your initialization parameter files.

- Obsolete Parameter

The `LM_PROCS` parameter became obsolete in release 8.1.6.

> **See Also:** Earlier editions of this manual for the version history on earlier releases that are no longer supported. (Release 8.0.3 and earlier).

Part I

Cluster Database Processing Fundamentals

Part One describes the fundamentals of Real Application Clusters processing. The chapters in this part are:

1 Introduction to Real Application Clusters

This chapter introduces cluster database processing and the features of Real Application Clusters.

In release 9*i*, **Real Application Clusters** embodies a breakthrough technology that offers significant advantages for **Online Transaction Processing (OLTP)**, **Decision Support System (DSS)**, and hybrid system types. With the increased functionality of Real Application Clusters, such systems can effectively exploit the redundancy of clustered environments.

You can also use Real Application Clusters to deliver high performance, increased throughput, and **high availability**. Before deploying Real Application Clusters, you must first understand how Real Application Clusters works, what resources it requires, and how to effectively deploy it.

This chapter includes the following topics:

Prerequisite Knowledge

Before reading about Real Application Clusters, you should be familiar with single **instance** Oracle processing.

Because you typically use Real Application Clusters in environments with more than one **node**, you have to configure at least two instances of Oracle so that they share access to an Oracle database. Having at least two instances means your administrative tasks are more complex than with single instance Oracle. For example, the way Oracle manages **rollback segments**, **redo log files**, **System Change Number (SCN)** generation, and so on, is more complex in Real Application Clusters than in single instance Oracle environments.

There are also additional architectural components in Real Application Clusters. These components synchronize data access within Real Application Clusters environments.

Real Applications Clusters Terminology

This manual assumes that you understand the terminology associated with single instance databases and client/server architecture. Each new term is shown in **bold face** type.

Note: Real Application Clusters terminology has changed considerably in Release 1 (9.0.1). Refer to the Glossary for new terms and definitions.

What is Real Applications Clusters?

Real Application Clusters is a computing environment that harnesses the processing power of multiple, interconnected computers. Real Application Clusters software and a collection of hardware known as a **cluster** unites the processing power of each component to become a single, robust computing environment. A cluster generally comprises two or more computers, (or *nodes)*. A cluster is also sometimes referred to as a loosely coupled computer system.

In Real Application Clusters environments, all nodes concurrently execute transactions against the same database. Real Application Clusters coordinates each node's access to the shared data to provide consistency and integrity.

Harnessing the power of multiple nodes offers obvious advantages. If you divide a large task into subtasks and distribute the subtasks among multiple nodes, then you can complete the task faster than if only one node did the work. Cluster database processing is clearly more efficient than sequential processing. It also provides increased performance for processing larger workloads and for accommodating growing user populations.

Real Application Clusters can effectively scale your applications to meet increasing data processing demands. As you add resources, Real Application Clusters can exploit them and extend their processing powers beyond the limits of the individual components.

You can use Real Application Clusters for many system types. For example, data warehousing applications accessing read-only data are prime candidates for Real Application Clusters. In addition, Real Application Clusters successfully manages increasing numbers of **Online Transaction Processing (OLTP)** systems as well as hybrid systems that combine the characteristics of both read-only and read / write applications.

Real Application Clusters also serves as an important component of robust **high availability** solutions. A properly configured Real Application Clusters environment can tolerate failures with minimal downtime.

Benefits of Real Applications Clusters

Some of the most important benefits beyond the obvious advantages of cluster database processing are described in the following sections. These benefits include improved throughput and **scalability** over single instance systems and improved response time. Real Application Clusters also provides an ideal high availability solution by resolving node failure in a clustered environment.

Real Application Clusters environments are functionally transparent when compared to single instance environments because they are functionally identical to single instance environments.

Scalability

Scalability is the ability to add additional nodes to Real Application Clusters and achieve markedly improved performance. Real Application Clusters can take advantage of additional equipment and harness the processing power of multiple systems.

High Availability

The term **high availability** refers to systems with redundant components that provide consistent, uninterrupted service, even in the event of hardware or software failures. In most high availability configurations, nodes are isolated from each other so a failure at one node does not affect the entire system. In such a case, surviving nodes compensate for the loss of the failed node through recovery and the system continues to provide data access to users. This means data is consistently available, more so than it would be with a single node upon node failure. High availability also implies increased database availability.

Transparency

The concept of **transparency** is the functional equivalence of single instance Oracle and shared configurations that use Real Application Clusters. Applications that run on single instance Oracle execute with the same results using Real Application Clusters. An Oracle database can be configured to execute in three different modes:

- Single instance exclusive
- Shared with a single instance
- Shared with two or more instances

High Performance Features of Oracle9*i* Real Application Clusters

Real Application Clusters takes advantage of the cluster database processing in a computer cluster without sacrificing Oracle's inherent transaction processing features.

The following subsections discuss Oracle features, both in exclusive and shared modes, that result in improved application performance when these applications run by using Real Application Clusters.

Buffer Cache Management Within a single instance, Oracle stores resources, such as data block information, in a buffer cache that resides in **memory**. Storing this information locally reduces the disk Input/Output (**I/O**) necessary for database operations. Since each node in Real Application Clusters has its own memory that is not shared with other nodes, Real Application Clusters must coordinate the buffer caches of different nodes while minimizing additional disk I/O that could reduce performance. The Oracle Global Cache Service technology maintains the high-performance features of Oracle while coordinating multiple buffer caches.

See Also: *Oracle9i Database Concepts* for detailed information about the buffer cache

Fast Commits, Group Commits, and Deferred Writes Fast commits, group commits, and deferred writes operate on each instance in Oracle and work the same whether in exclusive or shared mode.

Oracle only reads data blocks from disk if they are not already in the buffer caches of one of the instances. Because data block writes are deferred, they often contain modifications from multiple transactions.

Optimally, Oracle writes modified data blocks to disk only when necessary:

- When the blocks have not been used recently and new data requires buffer cache space (in shared or exclusive mode)
- During checkpoints (in shared or exclusive mount mode)
- When another instance needs the blocks (only in shared mode)
- Cache copies of **dirty block**s (changed blocks) across the interconnect (write / write consistency).

Note: Cache Fusion functionality allows direct memory writes of dirty blocks to alleviate the need to **forced disk write** and re-read (or **ping**) the committed blocks.

Row Locking and Multiversion Read Consistency Oracle's **row** locking feature allows multiple transactions from separate nodes to lock and update different rows of the same data block. This is done without any of the transactions waiting for the others to commit. If a row has been modified but not yet committed, then the original row values are available to all instances for read access. This is called *multiversion read consistency.*

Online Backups and Archiving Real Application Clusters supports all Oracle backup features that are available in exclusive mode, including both online and offline backups of either an entire database or individual tablespaces.

If you operate Oracle in `ARCHIVELOG` mode, then each online redo log file is made into an **archive (ARCH)** file before it is overwritten. In Real Application Clusters, each instance can automatically archive its own redo log files or one or more instances can manually archive the redo log files for all instances.

In `ARCHIVELOG` mode, you can make both online and offline backups. If you operate Oracle in `NOARCHIVELOG` mode, then you can only make offline backups. If you cannot afford any data loss, then Oracle strongly recommends that you operate your production databases in `ARCHIVELOG` mode.

2
Real Application Clusters Architecture

This chapter describes the architectural components used in **Oracle Real Application Clusters** processing. These components are used in addition to the components for single instances. They are thus unique to Real Application Clusters. Some of these components are supplied with Oracle database software and others are vendor-specific.

Topics in this chapter include:

See Also: *Oracle9i Database Concepts* for more information about Oracle's single instance architectural components

Overview of Clustered Systems Components

This section is an overview of Real Application Clusters, and how it uses **Cache Fusion**. Cache Fusion is a *diskless* **cache coherency** mechanism. Cache Fusion provides copies of blocks directly from the holding instance's memory cache to the requesting instance's memory cache.

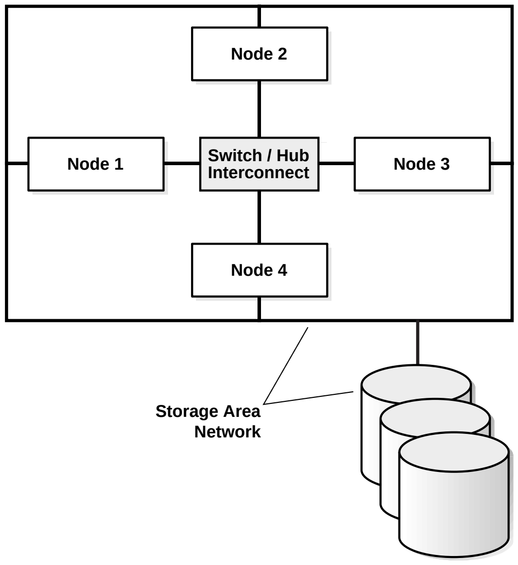
Real Application Clusters Software Components

Each hardware vendor implements cluster database processing by using **Operating System-Dependent (OSD) Layer**s. These layers provide communication links between the **Operating System** and the Real Application Clusters software described in this chapter.

Overview of Components for Clustered Systems

Clustered systems are composed of several components that synchronize operations among multiple **node**s accessing a shared database. A high-level view of these components in the Cache Fusion architecture appears in Figure 2–1.

Figure 2–1 Cluster Components for Cluster Database Processing



Real Application Clusters has these components:

- Cluster Manager
- The Global Cache Service and Global Enqueue Service
- Cluster Interconnect and Interprocess Communication (Node-to-Node)
- Disk Subsystems

The **Cluster Manager (CM)** oversees internode messaging that travels over the **interconnect** to coordinate internode operations. The **Global Cache Service (GCS)** oversees the operation of GCS resource and **Global Enqueue Service (GES)** enqueues. The cluster interconnect serves as a messaging pathway among the **node**s, and the shared disk drivers control access to the shared disk subsystems. The following section describes these components in more detail.

Cluster Manager

The Cluster Manager provides a global view of the cluster and all nodes in it. It also controls **cluster** membership. Typically, the Cluster Manager is a vendor-supplied component. However, Oracle supplies the Cluster Manager for Windows NT environments.

Real Application Clusters also cooperates with the Cluster Manager to achieve **high availability**. Working in conjunction with the Cluster Manager is the **Global Services Daemon (GSD)**. This **daemon** receives requests from the Real Application Clusters Control (**SRVCTL**) utility to execute administrative job tasks, such as **Startup (START)** or shutdown. The command is executed locally on each node, and the results are sent back to SRVCTL.

Failure Detection

A Cluster Manager disconnect can occur for three reasons. The client:

- Disconnects voluntarily
- Terminates the **process**
- Shuts down the node or the node fails

This is true even if one or more nodes fail. If the Cluster Manager determines that a node is inactive or not functioning properly, the Cluster Manager terminates all processes on that node or instance.

If there is a failure, recovery is transparent to user applications. The Cluster Manager automatically reconfigures the system to isolate the failed node and then notifies the Global Cache Service of the status. Real Application Clusters then recovers the database to a valid state.

The Node Monitor

The Cluster Manager includes a subset of functionality known as a *node monitor*. The Node Monitor polls the status of various **resource**s in a cluster including nodes, interconnect hardware and software, shared disks, and Oracle instances. The way that the Cluster Manager and its node monitor performs these operations is based on Oracle's implementation of the **Operating System-Dependent (OSD) Layer**.

The Cluster Manager informs clients and the Oracle server when the status of resources within a cluster change. For example, Real Application Clusters must know when another database instance registers with the Cluster Manager or when an instance disconnects from it.

As mentioned, the Cluster Manager monitors the status of various cluster resources, including nodes, networks and instances. The Node Monitor also serves the Cluster Manager by:

- Providing the basic node management interface modules needed by the Real Application Clusters software
- Discovering and tracking the membership state of nodes by providing a common view of cluster membership across the cluster
- Running on all nodes and monitoring the topology of the cluster by querying all nodes for their current membership
- Detecting changes in the state of active nodes signaling those events, diagnosing the changes, and coordinating a new common and consistent state among all nodes
- Notifying Real Application Clusters of the cluster membership changes

See Also: Chapter 10 for more information on high availability

Diagnosability Daemon

When you start Real Application Clusters instances, the alert log file shows the activity of various background processes. One of these is the diagnosability daemon. This daemon captures diagnostic data on instance process failures. No user control is required for this daemon. The diagnosability daemon should *not* be disabled or removed.

The Global Cache Service and Global Enqueue Service

The Global Cache Service and Global Enqueue Service are integrated components of Real Application Clusters that coordinate simultaneous access to the shared database and to shared resources within the database. They do this to maintain consistency and data integrity. This section describes the following features of the Global Cache Service and Global Enqueue Service:

- Transparency in the Global Cache Service and Global Enqueue Service
- Distributed Architecture in the Global Cache Service and Global Enqueue Service
- Fault Tolerance in the Global Cache Service and Global Enqueue Service
- Resource Mastering in the Global Cache Service and Global Enqueue Service
- Example of Global Cache Service Processing
- Interaction of the Global Cache Service and Global Enqueue Service with the Cluster Manager

Transparency in the Global Cache Service and Global Enqueue Service

The coordination of access to resources that is performed by the Global Cache Service and Global Enqueue Service are transparent to applications. Applications continue to use the same concurrency mechanisms used in single instance environments.

Distributed Architecture in the Global Cache Service and Global Enqueue Service

The Global Cache Service and Global Enqueue Service maintain a **Global Resource Directory** to record information about resources and enqueues held on these resources. The Global Resource Directory resides in memory and is distributed throughout the cluster to all nodes. In this distributed architecture, each node participates in managing global resources and manages a portion of the Global Resource Directory. This distributed resource management scheme provides fault tolerance and enhanced runtime performance.

Fault Tolerance in the Global Cache Service and Global Enqueue Service

The Global Cache Service and Global Enqueue Service have **fault tolerance**. They provide continual service and maintain the integrity of the Global Resource Directory even if multiple nodes fail. The shared database is accessible as long as at least one instance is active on that database after recovery is completed.

Fault tolerance also enables Real Application Clusters instances to start and stop at any time, in any order. However, instance reconfiguration can cause a brief delay.

Resource Mastering in the Global Cache Service and Global Enqueue Service

The Global Cache Service and Global Enqueue Service maintain information about a resource on all nodes that need access to it. The Global Cache Service and Global Enqueue Service usually nominate one node to manage *all* information about a resource.

Real Application Clusters uses a **resource mastering** scheme in which a resource is assigned based on its resource name to one of the instances that acts as the master for the resource. This results in an even and arbitrary distribution of resources among all available nodes. Every resource is associated with a master node.

The Global Cache Service and Global Enqueue Service optimize the method of resource mastering to achieve better load distribution and to speed instance startup. The method of resource mastering affects system performance during normal runtime activity as well as during instance startup. Performance is optimized when a resource is mastered locally at the instance that most frequently accesses it.

Resource Affinity in the Global Cache Service and Global Enqueue Service

The Global Cache Service and Global Enqueue Service employ **resource affinity**. Resource affinity is the use of **dynamic resource remastering** to move the location of the resource masters for a database file to the instance where resource operations are most frequently occurring. This optimizes the system in situations where update transactions are being executed on one instance. When activity shifts to another instance the resource affinity will correspondingly move to the new instance. If activity is not localized, the resource ownership is distributed to the instances equitably. Dynamic resource remastering is the ability to move the ownership of a resource between instances of Real Application Clusters during runtime and without affecting availability. It can be used to defer resource mastering when new instances enter the cluster. Resource remastering is deferred when instances enter or leave the cluster.

Example of Global Cache Service Processing

Assume that a node in a cluster needs to modify block number *n* in the database. At the same time, another node needs to update the same block to complete a transaction.

Without the Global Cache Service, both nodes would simultaneously update the same block. With the Global Cache Service, only one node can update the block; the other node must wait. The Global Cache Service ensures that only one instance has the right to update a block at any given time. This provides data integrity by ensuring that all changes made are saved in a consistent manner.

Interaction of the Global Cache Service and Global Enqueue Service with the Cluster Manager

The Global Cache Service and Global Enqueue Service operate independently of the Cluster Manager. They rely on the Cluster Manager for timely and correct information about the status of other nodes. If the Global Cache Service and Global Enqueue Service cannot get the information they need from a particular instance in the cluster, then they will shut down the instance that is out of communication. This ensures the integrity of Real Application Clusters databases, as each instance must be aware of all other instances to coordinate disk access.

See Also: Chapter 5 and Chapter 6 for more information on interinstance coordination and details about how the Global Cache Service and Global Enqueue Service control data access among instances

Cluster Interconnect and Interprocess Communication (Node-to-Node)

Real Application Clusters derives most of its functional benefits from its ability to run on multiple interconnected machines. Real Application Clusters relies heavily on the underlying high speed **interprocess communication (IPC)** component to facilitate this.

The IPC defines the protocols and interfaces required for the Real Application Clusters environment to transfer messages between instances. Messages are the fundamental units of communication in this interface. The core IPC functionality is built around an asynchronous, queued messaging model. IPC is designed to send and receive discrete messages as fast as the hardware allows. With an optimized communication layer, various services can be implemented above it. This is how the Global Cache Service and the Cluster Manager perform their communication duties.

Disk Subsystems

In addition to the OSD layers, Real Application Clusters also requires that all nodes must have simultaneous access to the disks. This gives multiple instances concurrent access to the same database.

3
Cluster Hardware Architecture

This chapter describes the hardware components and various high-level architectural models that typify cluster environments. It explains the basic hardware for nodes as well as the hardware that unites the individual nodes into a cluster.

Topics in this chapter include:

- Overview of Cluster Hardware Components
- Node Components
- Memory, Interconnect, and Storage
- The High Speed IPC Interconnect
- Storage Access in Clustered Systems
- Clusters: Nodes and the Interconnect
- Interoperability with Other Systems

Overview of Cluster Hardware Components

A cluster comprises two or more nodes that are linked by an interconnect. The interconnect serves as the communication path between the **nodes** in the cluster. The Oracle **instances** use the interconnect for the messaging required to synchronize each instance's manipulation of the shared data. The shared data that the nodes access resides in **storage** devices.

The architectural model you select to deploy your Real Application Clusters application depends on your processing goals. This chapter describes cluster components in more detail.

Node Components

A node has these main components:

- **central processing unit** – The main processing component of a computer that reads from and writes to the computer's main memory.
- **memory** – The component used for programmatic execution and the buffering of data.
- **interconnect** – The communication link between the nodes.
- **storage** – A device that stores data. This is usually persistent storage that must be accessed by read / write transactions to alter its contents.

You can purchase these components in several configurations. The arrangement of these components determines how each node in a cluster accesses memory and storage.

Memory, Interconnect, and Storage

All clusters use CPUs in more or less the same way. However, you can configure the remaining components, memory, storage, and the interconnect in different ways for different purposes.

Memory Access

Multiple CPUs are typically configured to share main memory. This enables you to create a single computer system that delivers scalable performance. This type of system is also less expensive to build than a single CPU with equivalent processing power. A computer with a single CPU is known as a **uniprocessor**.

There are two configurations of shared memory systems discussed in this section:

- Uniform Memory Access (UMA)
- Non-Uniform Memory Access (NUMA)

Uniform Memory Access (UMA)

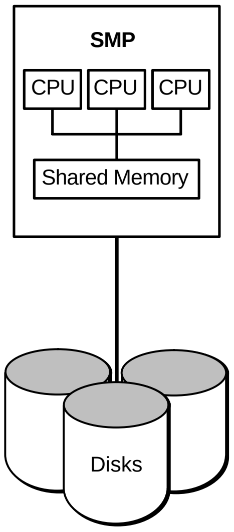
In **uniform memory access (UMA)** configurations, all processors can access main memory at the same speed. In this configuration, memory access is uniform. This configuration is also known as a **Symmetric Multi-Processor (SMP)** system as illustrated in Figure 3–1.

Non-Uniform Memory Access (NUMA)

non-uniform memory access (NUMA), means that all processors have access to all memory structures. However, the memory accesses are not equal. In other words, the access cost varies depending on what parts of memory each processor accesses. In NUMA configurations, the cost of accessing a specific location in main memory is different for some of the CPUs relative to the others.

Performance of UMA Versus NUMA Performance in both UMA and NUMA systems is limited by memory bus bandwidth. This means that as you add CPUs to the system beyond a certain point, performance will not increase linearly. The point where adding CPUs results in minimal performance improvement varies by application type and system architecture. Typically SMP configurations do not scale well beyond 24 to 64 processors.

Figure 3–1 Tightly Coupled Shared Memory System or UMA



Advantages of Shared Memory

The cluster database processing advantages of shared memory systems are:

- Memory access is less expensive than access in a loosely coupled system
- Shared memory systems are easier to administer than a cluster

A disadvantage of shared memory systems for cluster database processing is that scalability is limited by the bandwidth and latency of the bus and by available memory.

The High Speed IPC Interconnect

The high speed **interprocess communication (IPC)** interconnect is a high bandwidth, low latency communication facility that connects each node to the other nodes in the cluster. The high speed interconnect routes messages and other cluster database processing-specific traffic among the nodes to coordinate each node’s access to data and to data-dependent resources.

Real Application Clusters also makes use of user-mode IPC and *memory-mapped IPC*. These substantially reduce CPU consumption and reduce IPC latency.

You can use Ethernet, a **Fiber Distributed Data Interface (FDDI)**, or some other proprietary hardware for your interconnect. Also consider installing a backup interconnect in case your primary interconnect fails. The backup interconnect enhances high availability and reduces the likelihood of the interconnect becoming a single point-of-failure.

Storage Access in Clustered Systems

Clustered systems use several storage access models. Each model uses a particular resource sharing scheme that is best used for a particular purpose.

The type of storage access Real Application Clusters uses is independent of the type of memory access it uses. For example, a cluster of SMP nodes can be configured with either uniform or non-uniform disk subsystems.

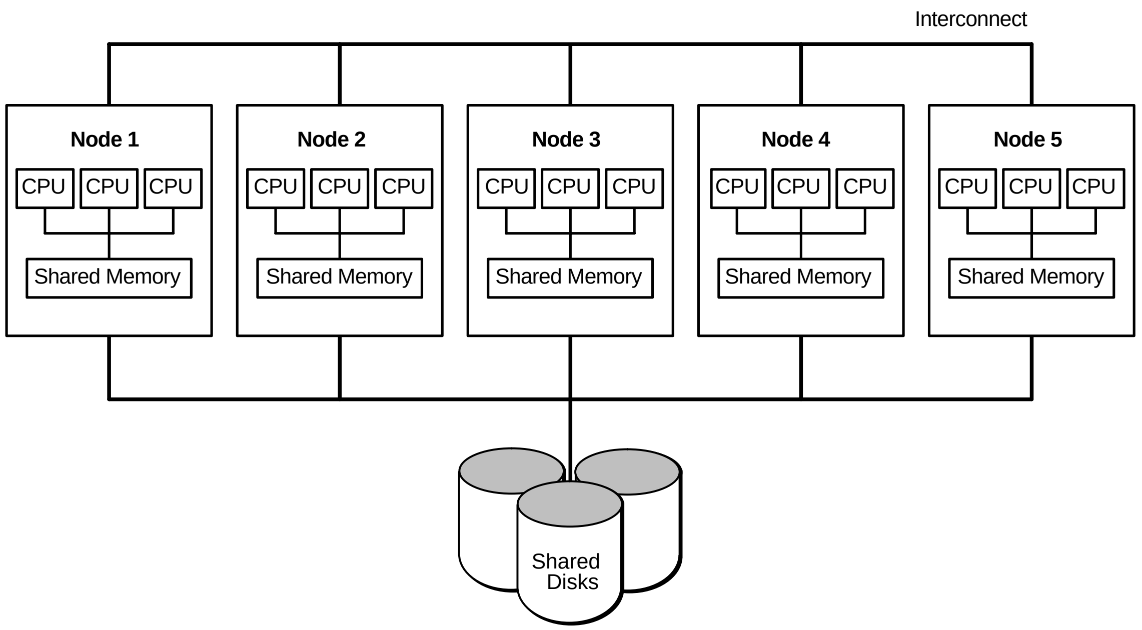
Real Application Clusters uses two storage access models discussed in these sections:

- Uniform Disk Access
- Non-Uniform Disk Access

Uniform Disk Access

In uniform disk access systems, or shared disk systems, as shown in Figure 3–2, the cost of disk access is the same for all nodes.

Figure 3–2 Uniform Access Shared Disk System



The cluster in Figure 3–2 is composed of multiple SMP nodes. Shared disk subsystems like this are most often implemented by using shared SCSI or Fibre Channel connections to a large number of disks.

Fibre Channel is a generic term for a high speed serial data transfer architecture recently standardized by the American National Standards Institute (ANSI). The Fibre Channel architecture was developed by a consortium of computer and mass storage manufacturers.

Advantages of Uniform Disk Access

The advantages of using cluster database processing on shared disk systems with uniform access are:

- High availability; all data is accessible even if one node fails
- Incremental growth

Non-Uniform Disk Access

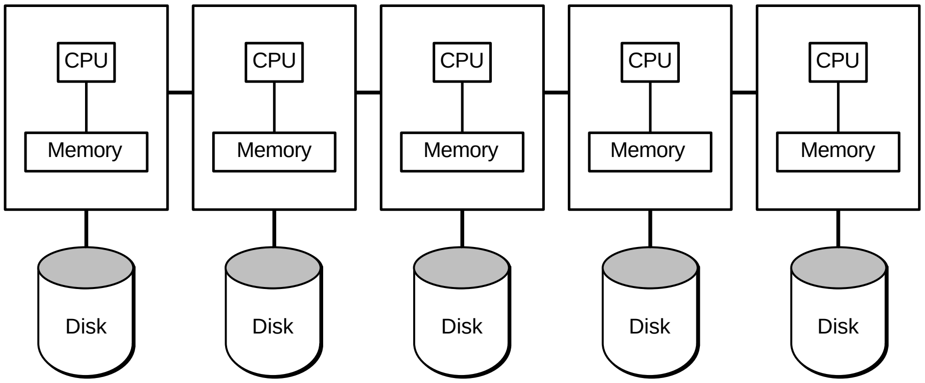
In some systems, disk storage is attached to only one node. For that node, the access is local. For all other nodes, a request for disk access or data must be forwarded by a software virtual disk layer over the interconnect to the node where the disk is locally attached. This means that the cost of a disk read or write varies significantly depending on whether the access is local or remote. The costs associated with reading or writing the blocks from the remote disks, including the interconnect latency and the IPC overhead, all contribute to the increased cost of this type of operation versus the cost of the same type of operation by using a uniform disk access configuration.

MPP Systems and Resource Affinity

Non-uniform disk access configurations are commonly used on systems known as *shared nothing* or **Massively Parallel Processing (MPP)** systems. If a node fails on a high availability system, then you can usually reconfigure local disks to be local to another node. For such non-uniform disk access systems, Real Application Clusters requires that the virtual disk layer be provided at the system level. In some cases it is much more efficient to move work to the node where the disk or other I/O device is locally attached rather than to use remote requests. This ability to collocate processing with storage is known as **resource affinity**. It is used by Oracle in a variety of areas including parallel execution and backup.

Figure 3–3 illustrates a shared nothing system:

Figure 3–3 Non-Uniform Disk Access



Advantages of Non-Uniform Disk Access

The advantages of using cluster database processing on MPP or non-uniform disk access systems are:

- The number of nodes is not limited by the disk connection hardware
- The total disk storage can be large due to the ability to add nodes

Clusters: Nodes and the Interconnect

As described previously, to operate Real Application Clusters you must use either a uniprocessor, SMP (UMA), or NUMA memory configuration. When configured with an interconnect, two or more of these types of processors make up a cluster. The performance of a clustered system can be limited by a number of factors. These include various system components such as the memory bandwidth, CPU-to-CPU communication bandwidth, the memory available on the system, the I/O bandwidth, and the interconnect bandwidth.

Interoperability with Other Systems

Real Application Clusters is supported on a wide range of clustered systems from a number of different vendors. The number of nodes in a cluster that Real Application Clusters can support is significantly greater than any known implementation. For a small system configured primarily for high availability, there might only be two nodes in the cluster. A large configuration, however, might have 40 to 50 nodes in the cluster. In general, the cost of managing a cluster is related to the number of nodes in the system. The trend has been toward using a smaller number of nodes with each node configured with a large SMP system that uses shared disks.

Part II

Resource Coordination in Real Application Clusters

Part Two describes the resource coordination that Real Application Clusters performs to synchronize data access and ensure data integrity. The chapters in this part are:

4 Local Resource Coordination

This chapter provides an overview of resource coordination and a description of local resources that are employed by the **Global Cache Service (GCS)** and **Global Enqueue Service (GES)**.

Topics in this chapter include:

- Overview of Resource Coordination
- Resource Coordination Components
- Local Concurrency Controls

See Also: Chapter 5, "Cache Fusion and the Global Cache Service" for information on the GCS, and Chapter 6, "Coordination by the Global Enqueue Service" for information on the Global Enqueue Service

Overview of Resource Coordination

There are three general categories of concurrency control used by the Real Application Clusters software:

- Independent resources
- Global Cache Service resources
- Global Enqueue Service enqueues

This chapter primarily discusses Independent resources.

Oracle uses these concurrency controls to maintain buffer cache coherency and perform other synchronization tasks. This guarantees database integrity in a clustered environment. To transfer data blocks among database caches, buffers are transferred by way of a high speed interconnect. Just as in a single instance system, disk writes are required for cache replacement.

Real Application Clusters synchronizes global resources among all active instances in a cluster. There are two main types of global resources:

- Resources used by the Global Cache Service for resource management.
- Global resources that Oracle synchronizes within a cluster to coordinate GES enqueues

Oracle uses the **Global Cache Service (GCS)**, **Global Enqueue Service (GES)**, and independent resources to maintain buffer cache coherency and perform other synchronization tasks. Together, they guarantee database integrity in a clustered environment.

Coordination of concurrent tasks within a **shared cache** server is called synchronization. Resources such as data blocks and **enqueue**s must be synchronized as **node**s within a cluster acquire and release their ownership. The synchronization provided by Real Application Clusters maintains cluster-wide concurrency of the resources and in turn ensures the integrity of the shared data.

The key to successful cluster database processing is to divide the tasks that require resources among the nodes so that very little synchronization is necessary. The less synchronization that is necessary, the better your system's **speed up** and **scale up**. The overhead of synchronization can be very expensive if excessive internode communication is necessary.

Synchronization among nodes is accomplished with the high-speed IPC interconnect linking the clustered processors. For cluster database processing within a node, messaging is not necessary; shared memory is used instead. Messaging and enqueuing among nodes is handled by the GES.

The amount of synchronization depends on the amount of resources and the number of users and tasks working on the resources. Little synchronization might be needed to coordinate a small number of concurrent tasks, but many concurrent tasks can require significant synchronization.

Real Application Clusters employs a **Global Resource Directory** distributed across all instances in a **cluster**. Resources are mastered at one instance. The GCS and GES manage the resources that they master.

> **See Also:** Chapter 2, "Real Application Clusters Architecture"for information on resource mastering and resource affinity. See Chapter 10, "High Availability Concepts and Best Practices" for information on resource remastering in node failover

Resource Coordination Components

The components within a **cluster database** that require concurrency control include:

- Row level access
- Block level access
- Space Management
- System Change Number (SCN) creation
- Data dictionary cache
- Library cache

Local Concurrency Controls

There are three types of local concurrency controls: latches, enqueues, and row locks. Latches do not affect the global operations of clustered environments. Enqueues, however, can be both local to an instance and global to a cluster. While locking is automated in Real Application Clusters, it does employ independent row locks.

Latches

A **latch** is a simple, low level serialization mechanism that protects a critical resource. All latches are local. Latches do not protect datafiles, they are automatic, and they are held for a very short duration. Latches are either in use or free. Each critical resource has its own latch. As mentioned, because latches are synchronized within a node, they do not facilitate internode synchronization.

Local Enqueues

A local **enqueue** is a shared memory structure that serializes access to database resources. Enqueues are local to one instance when Real Application Clusters is not enabled. However, when Real Application Clusters is enabled, enqueues can be global to a database. Enqueues are associated with a session or transaction. Oracle can use enqueues in any of three modes:

- null (N) mode
- shared (S) mode
- exclusive (X) mode

Enqueues are held longer than latches, have finer granularity, have more modes than latches, and protect more database resources. For example, if you request **table lock**, or a **Database Mount Lock**, your request is assigned an enqueue.

> **See Also:** *Oracle9i Database Reference* for descriptions of all GES enqueues

Row Locks

Row locks are locks that protect selected rows. The following statements create row locks:

- `INSERT`
- `UPDATE`
- `DELETE`
- `SELECT` with the `FOR UPDATE` clause

These locks are stored in the block, and each lock refers to the global transaction lock.

As in single instance Oracle, Real Application Clusters controls concurrency down to the row level. (The finest lock granularity is at the row level.) This is a key advantage of Real Application Clusters over competing parallel processing systems. To keep the cache coherent, access to the data blocks is controlled by the GCS.

GCS resources and row locks operate independently of the GCS. An instance can disown a GCS resource without affecting row locks held in the set of blocks managed by the GCS. A row lock is acquired during a transaction. During a transaction, the GCS can transfer blocks between instances, independently of when transactions commit.

In contrast, transactions do not release row locks until changes to the rows are either committed or rolled back. Oracle uses internal mechanisms for concurrency control. These isolate transactions so modifications to data made by one transaction are not visible to other transactions until the transaction modifying the data commits. The row lock concurrency control mechanism is independent of the GCS. Concurrency control does not require GCS resources, and GCS resource operations do not depend on individual transactions committing or rolling back.

5

Cache Fusion and the Global Cache Service

This chapter is an overview **Cache Fusion** and how the **Global Cache Service (GCS)** operates.

Topics in this chapter include:

See Also: Chapter 6, "Coordination by the Global Enqueue Service" for information on the Global Enqueue Service

Overview of Cache Fusion

Cache Fusion is a new technology that uses a high speed **interprocess communication (IPC)** interconnect to provide cache to cache transfers of data blocks between **instance**s in a cluster. This eliminates disk I/O (which is inherently slow, since it is a mechanical process) and optimizes read/write concurrency. Block reads take advantage of the speed of IPC and an interconnecting network. Cache Fusion also relaxes the requirements of data partitioning.

Cache Fusion addresses these types of concurrency between instances, each of which is discussed in the following sections:

- Concurrent Reads on Multiple Nodes
- Concurrent Reads and Writes on Different Nodes
- Concurrent Writes on Different Nodes

Concurrent Reads on Multiple Nodes

Concurrent reads on multiple nodes occur when two instances need to read the same data block. Real Application Clusters easily resolves this situation because multiple instances can share the same blocks for read access without cache coherency conflicts.

Concurrent Reads and Writes on Different Nodes

Concurrent reads and writes on different nodes are the dominant form of concurrency in **Online Transaction Processing (OLTP)** and hybrid applications. A read of a data block that was recently modified can be either for the current version of the block or for a read-consistent previous version. In both cases, the block will be transferred from one cache to the other.

Concurrent Writes on Different Nodes

Concurrent writes on different nodes occur when the same data block is modified frequently by processes on different instances.

The main features of the cache coherency model used in Cache Fusion are:

- The cache-to-cache data transfer is done through the high speed IPC interconnect. This virtually eliminates any disk I/Os to achieve cache coherency.
- The **Global Cache Service (GCS)** tracks one or more **past image (PI)** for a block in addition to the traditional GCS resource roles and modes. (The GCS tracks

blocks that were shipped to other instances by retaining block copies in memory. Each such copy is called a **past image (PI)**. In the event of a failure, Oracle can reconstruct the current version of a block by using a PI.

- The work required for recovery in node failures is proportional to the number of failed nodes. Oracle must perform a log merge in the event of failure on multiple nodes.
- The number of context switches is reduced because of the reduced sequence of round trip messages. In addition, **database writer (DBWR)** is not involved in Cache Fusion block transfers. Reducing the number of context switches adds to the more efficient use of the cache coherency protocol.

Global Cache Service Operations

The GCS tracks the location and status (mode and role) of data blocks, as well as the access privileges of various instances. Oracle uses the GCS for cache coherency when the current version of a data block is in one instance's buffer cache and another instance requests that block for modification. It is also used for reading blocks.

Following the initial acquisition of exclusive resources in subsequent transactions multiple transactions running on a single Real Application Clusters instance can share access to a set of data blocks without involvement of the GCS as long as the block is not transferred out of the local cache. If the block has to be transferred out of the local cache, then the **Global Resource Directory** is updated by the GCS.

Cache Coherency

Data blocks are the most often required database resources. The GCS manages all types of data blocks.

The GCS ensures **cache coherency** by requiring that instances acquire a resource cluster-wide before modifying or reading a database block. Thus, the GCS synchronizes global cache access, allowing only one instance at a time to modify a block. The GCS coordination of the buffer caches located on separate nodes provides cache coherency to Real Application Clusters. The GCS ensures the status of data blocks cached in any mode in the cluster is globally visible and maintained.

Oracle's multi-versioning architecture distinguishes between current data blocks and one or more **consistent read (CR)** versions of a block. A data block can reside in many buffer caches under shared resources. The current block contains changes for all committed and yet-to-be-committed transactions. A **consistent read (CR)** version of a block represents a consistent snapshot of the data at a previous point in time. Consistent read versions are produced by applying rollback segment information. Both current and consistent read blocks are managed by the GCS

To transfer data blocks among database caches, buffers are shipped by means of a high speed IPC interconnect. Just as in a single instance system, disk writes are only required for cache replacement. A past image (PI) of a block is kept in memory before the block is sent and if it is a **dirty block**. In the event of failure, Oracle can reconstruct the current version of the block by reading PIs.

GCS Resource Modes and Roles

GCS resources track the transmission of blocks through the system. The same block can exist in multiple caches as a result of block transfers. The block can be held in different modes depending on whether a resource holder intends to modify data or merely read them.

It is important to understand that a resource is identified by these factors:

- Resource mode: The modes are null, shared, and exclusive.
- Resource role: The roles are local and global

Resource Modes

Resource modes are generally determined by the holder, as part of a request for a data block. The **resource mode**s determine whether the holder can modify the block. Table 5–1 compares the **null (N) mode**, **shared (S) mode**, and **exclusive (X) mode**.

Table 5–1 Global Cache Service Resource Modes

Resource Mode	Identifier	Description
Null	N	Holding a resource at this level conveys no access rights.
Shared	S	A protected read. When a resource is held at this level, a process cannot modify it. Multiple processes can read the resource.
Exclusive	X	When a resource is held at this level, it grants the holding process exclusive access. Other processes cannot write to the resource. Consistent reads of older blocks are still available.

Resource Roles

Oracle assigns GCS **resource role**s to the holder. They supplement the user-requested modes based on the knowledge of the global state of the blocks by the resource management system. The roles are either local or global:

- When a block is first read into an instance's cache and other instances have not read the block, the block is said to be locally managed. It is therefore assigned a *local* role.
- After the block has been modified by the local instance and transmitted to another instance, it is considered globally managed. It is therefore assigned a *global* role.

All GCS resources effectively have the local role if they only exist in one cache. Roles are mutually exclusive. If a data block was changed in one instance and subsequently transferred to another instance, the buffer containing the data is considered globally dirty. That is, the resource has the global role.

When running a single instance in exclusive mode, all concurrency control is done within the instance. With Real Application Clusters in shared mode, synchronization is accomplished by the GCS or **Global Enqueue Service (GES)**.

Past Images

A block is initially acquired in local role, with no past images (PIs) present. Only after a block has been changed (or becomes a **dirty block**) and another instance requests it, does the node that dirtied the block begin to keep PIs. The resource then becomes global.

The exclusive current copy of a data block can only exist in the cache of the instance that last modified it. There might also be PIs of the block in other caches. These PIs represent earlier versions of the block with modifications that have not been written to disk, and can be used for consistent reads in the cluster.

Write Protocol and Past Image Tracking

When a block is requested for modification for a current read, the instance that last modified a data block sends the block by using a high speed IPC interconnect and retains a PI. Writes to disks are only triggered by cache replacements and checkpoints. The write protocol is largely asynchronous. This reduces the I/O requirements of an Real Application Cluster node to those comparable to a single instance.

Consider when an instance intends to initiate a write of a data block, and the resource has a global role, and it does not have the current buffer, only a PI. Under these circumstances the instance informs the GCS. The GCS then forwards the write request to the instance where the current (or most recent) version of the block is held.

The holder of the current version writes the block to disk. Then, upon completion, the holder sends a completion message to the GCS. Finally, all instances with PI buffers for the written block free their PI buffers.

The GCS always mediates global operations at the cache layer and tracks the latest global state of resources.

Real Application Clusters Resource Control Mechanisms

To guarantee coherent and accurate access to cached data, the cluster database controls access to shared resources. This includes resources such as data blocks or data structures used for other purposes such as instance management, data dictionary access, and recovery synchronization.

When Oracle reads a data block into memory, Oracle opens a GCS resource to coordinate concurrent access to the resource from multiple instances. Oracle opens or converts the resource in different modes and roles depending on whether:

- The data accessed is to be modified or read
- A data block exists in the cache of only one instance or in multiple caches

Oracle closes GCS resources when the block access mode is down-converted to NULL, and there no PI, or when Oracle flushes the buffer from the cache due to cache replacement.

By default, a resource is allocated for each data block in a cache. Due to Cache Fusion and the virtual elimination of immediate disk writes that occur when other instances make modification requests, the performance overhead of concurrency on shared data between instances is diminished. This reduces the tuning and administrative effort for Real Application Clusters environments.

Note: Cache Fusion only works with the default resource control scheme. If you override Cache Fusion and set `GC_FILES_TO_LOCKS` in your initialization parameter file and assign resources to multiple blocks, then Oracle uses pre-9.0.1 behavior. In other words, Oracle will use **forced disk write**s for cross-instance modification requests. This is not recommended in most circumstances.

Eliminating the Need for Configuring Resources

The new architecture for global resource control and Oracle's breakthrough Cache Fusion technology simplify the performance tuning and administration of Real Application Clusters environments. The importance of configuring accurate resource allocations to provide optimal performance, as well as the planning of sufficient capacity for Global Cache Service and **Global Enqueue Service (GES)** resources has been largely reduced. If you use the default resource control scheme, you do not need special initialization parameter settings to configure resources in Real Application Clusters.

Resource Control, Cache-to-Cache Transfer, and Cache Coherency

The GCS assigns and opens resources for each database block read into the buffer cache. Oracle closes resources when the resources do not manage any more buffers or when buffered blocks are written to disks due to cache replacement and free buffer requests.

When Oracle closes a resource, it returns it to a free list from which Oracle can assign new resources. The size of the free list is by default equal to the size of the buffer cache. Oracle allocates the free list from the shared pool.

There are no special considerations for global enqueues. Their number is calculated automatically at startup and Oracle records the calculated values in the `alert.log` file. You do not need to set initialization parameters.

Generally, global enqueues have different uses and semantics than GCS resources. Global enqueues are used by the different kernel layers such as the row cache, the library cache and so on, to coordinate access to a variety of objects.

Cache Fusion Resource Assignment and Block Coverage

This section describes how Cache Fusion controls resource assignments. The topics in this section are:

- Block Access Modes and Buffer States
- How Buffer States and Block Access Modes Change
- Block Access Modes Can Be Compatible or Incompatible

Block Access Modes and Buffer States

There are three concurrency control concepts that need to be distinguished: buffer state, resource mode, and resource role:

- The **buffer state** is the state of a buffer in the local cache of an instance.
- The **resource mode** controls global access rights for instances in a cluster.
- The **resource role** defines whether a block is cached in only one instance (local) or if it cached in multiple instances (global).

The buffer state of a block relates directly to the access mode of the block and the role assigned to the instance in relation to the block. For example, if a buffer is in **exclusive current (XCUR)** state, you know that an instance owns the resource in exclusive mode. In addition, if the data block is read from disk and cached in only one instance, the role is *local*.

There can be only one block buffered in XCUR state in the cluster at any time. To perform modifications on a block, a process must assign an XCUR buffer state to the buffer containing the data block.

If another instance requests reading the same block in its most current version, for example, then Oracle changes the access mode from exclusive to shared, sends the block and keeps a PI buffer if the buffer contained a **dirty block**. It sends a current read version of the block to the requesting instance. At this point, the first instance has the current block, the changes made to it, and the requesting instance also has the current block in shared mode. The role of the resource becomes global. There can be multiple **shared current (SCUR)** versions of this block cached at any time.

Finding the State of a Buffer

To see a buffer's state, query the STATUS column of the V$BH dynamic performance view. The V$BH view provides information about each buffer header as shown in Table 5–2.

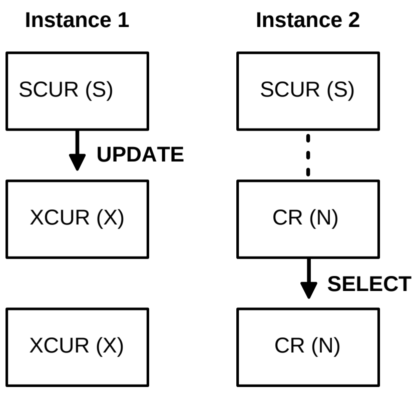
Table 5–2 Block Access Modes and Buffer States

Block Access Mode	Buffer State Name	Description
X	XCUR	Instance has exclusive access to the block and can modify it.
S	SCUR	Instance has shared access to the block and can only perform reads.
NULL	CR	Instance can perform a consistent read of the block. (That is, if it contains an older version of the data.).

How Buffer States and Block Access Modes Change

Figure 5–1 shows how buffer states and block access modes change as instances perform various operations on a given buffer. The block access mode appears in parentheses.

Figure 5–1 How Buffer States and Block Access Modes Change



In Figure 5–1, the two instances begin with blocks in shared current mode and with shared resources. When Instance 1 performs an update on the block, its access mode on the block changes to exclusive mode (X). The shared resource owned by instance 2 converts to null mode (N). Meanwhile, the block state in instance 1 becomes XCUR, and in instance 2 it becomes CR. These block access modes are compatible.

Block Access Modes Can Be Compatible or Incompatible

When one process owns a resource in a given mode, another process requesting a resource in any particular mode succeeds or fails as shown in Table 5–3.

Table 5–3 Block Access Mode Compatibility

Mode Requested:	**Null**	**S**	**X**
Mode Owned			
Null	Succeed	Succeed	Succeed
S	Succeed	Succeed	Fail
X	Succeed	Fail	Fail

Cache Fusion Scenarios

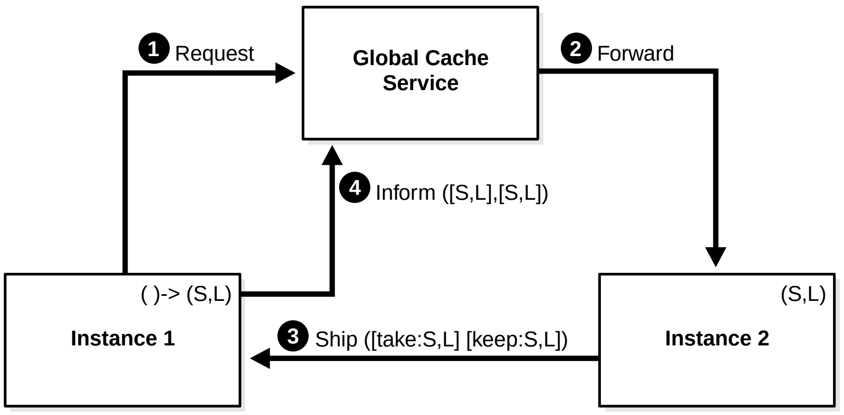
The following scenarios illustrate the key points of Cache Fusion processing. These scenarios, which illustrate key concepts and do not address all possible configurations, are described in the following sections:

- Requesting a Block for a Read from Another Instance: Scenario
- Requesting a Changed Block for Modification: Scenario
- Writing Blocks to Disk: Scenario

Requesting a Block for a Read from Another Instance: Scenario

The scenario shown in Figure 5–2 assumes that one instance has read a data block into its cache. The data block is protected by a resource in shared mode (S) and its role is local (L). This indicates that the block only exists in the local cache of this instance.

Figure 5–2 Requesting a Block for a Read from Another Instance



S	=	Shared
L	=	Local
( )	=	Null
(x,y)	=	Held
x	=	mode
y	=	role
([x,y], [x,y])	=	Message Content ([requestor disposition], [holder disposition])

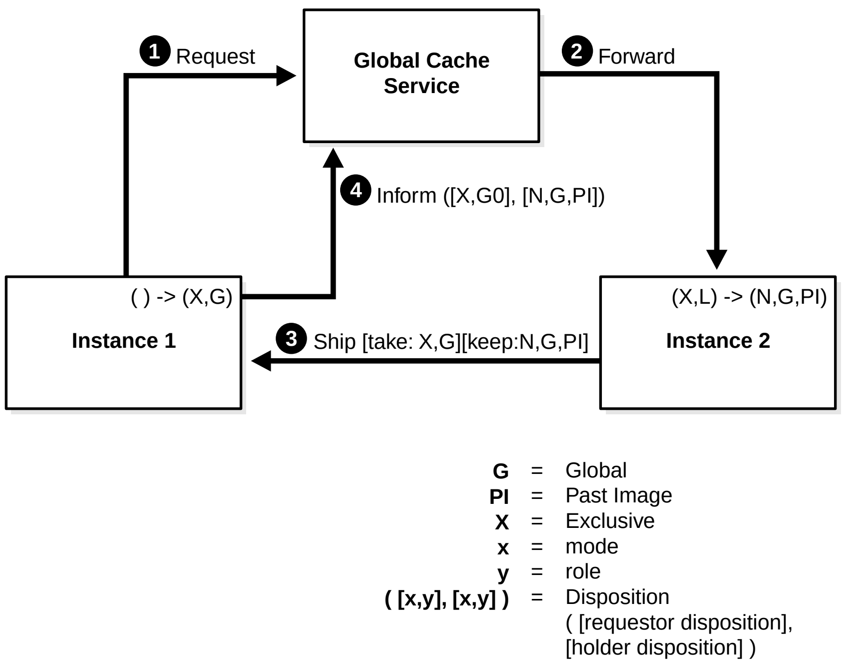
1. Instance 1 submits a request to the GCS to read a block. The GCS always knows the global distribution of resources, so it knows that a copy of the block is already in the cache of Instance 2.
2. The GCS then forwards the request to Instance 2.
3. The holding instance (Instance 2) transmits a copy of the block to the requesting instance (Instance 1), but keeps the resource in shared mode and also retains the local role. Along with the block, Instance 2 transmits its own resource disposition (shared and local), and the mode and role the requestor is to use in taking the resource. The mode is shared and role is local.
4. Once Instance 1 has received the block, it informs the GCS that it has taken the block and resource in shared mode and local role, and that the sender has retained the block and resource with the same disposition.

Note that the block and the mode and role information is transferred cache-to-cache through the high speed IPC interconnect without any disk I/O.

Requesting a Changed Block for Modification: Scenario

The scenario shown in Figure 5–3 assumes that the data block has been changed (or *dirtied*) by one instance and held in exclusive mode (X). Furthermore, this scenario assumes that the block has only been accessed by the instance that changed it. That is, only one copy of it exists cluster-wide. In other words, the block is in a local role (L).

Figure 5–3 Requesting a Changed Block for Modification



1. As in the first scenario, the instance attempting to modify the block (Instance 1) submits a request to the GCS.
2. The GCS transmits the request on to the holder (Instance 2).
3. Instance 2 receives the message, sends the block to Instance 1. Before sending the block, the resource is downgraded to null mode and keeps the changed (dirty) buffer is kept as a PI. Thus, the role changes to global (G), because the

block is dirty. Along with the block, Instance 2 relays that to the requestor it retained a PI copy and a null resource. In the same message, it also specifies that the requestor take the block held in exclusive mode and with a global role.

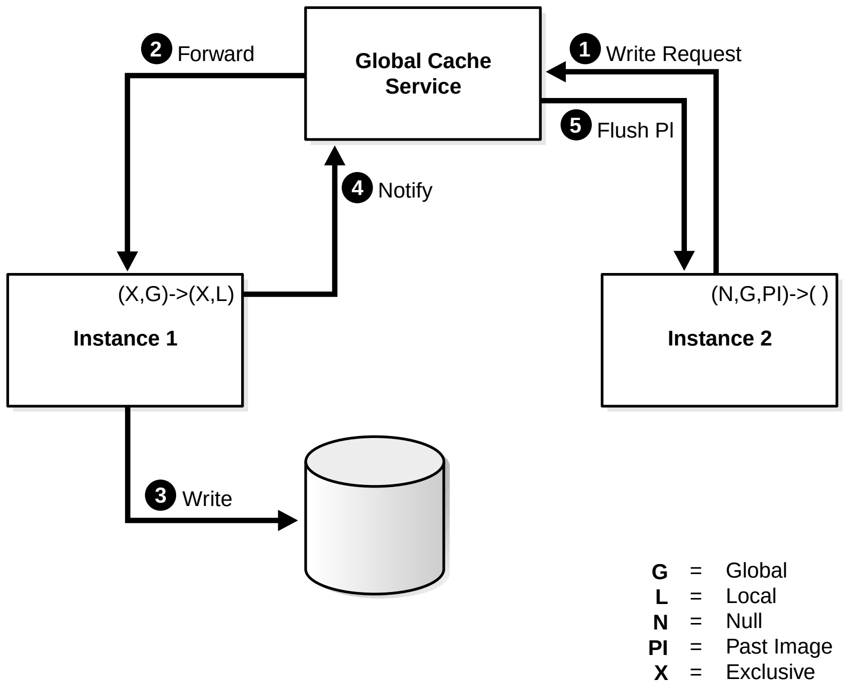
4. On receipt of the block and the resource dispositions, Instance 1 informs the GCS that it is now holding the block in exclusive mode and with a global role. Meanwhile, Instance 2 (the former holder) retains a PI of the same block in null mode and global role. Note that the data block is not written to disk before the resource is granted to the other instance. That is, DBWR is not involved in the cache coherency scheme.

Writing Blocks to Disk: Scenario

The scenario shown in Figure 5–4 illustrates how an instance can **checkpoint** at any time or replace buffers in the cache due to free buffer requests. Because multiple versions of the data block with changes could exist in the caches of instances in the cluster, a write protocol mediated by the GCS must ensure that the current version of the data is written to disk. It must also ensure that all existing previous versions are purged from the other caches. A write request for a data block can originate in any instance that has the current or previous version of the block.

In this scenario, assume that the instance holding a PI buffer in null mode requests that the buffer be written.

Figure 5–4 Writing Blocks to Disk



1. Instance 2 first sends a write request to the GCS.
2. The GCS forwards the request to Instance 1 (the current block holder). The GCS remembers that a write at the **System Change Number (SCN)** is pending. The GCS also remembers that it has to notify nodes that have PIs of the same block.
3. Instance 1 receives the write request and writes the block to disk.
4. Instance 1 logs the completion of the write. It then notifies the GCS of the write completion. Instance 1 also informs the GCS that the resource role can become local because Instance 1 performed the write of the current block. After completion of the protocol, all PIs of the block should be discarded.
5. After receipt of the notification, the GCS orders all PI holders to discard (or *flush*) their PIs. Discarding in this case means that on receipt of the message, PI holders log that the current block has been written and the buffer is released. The PI is no longer needed for recovery. The buffer is essentially free and the resource previously held in null mode is closed.

See Also: *Oracle9i Real Application Clusters Deployment and Performance* for additional information on System Change Numbers

How the GCS Grants and Coordinates Resource Requests

This section describes the basic concepts of how the GCS grants and coordinates resource requests. The topics in this section are:

- Interrupt and Completion Processing
- Block Access Requests are Queued
- How the GCS Grants and Coordinates Resource Requests
- Block Access Requests are Granted and Converted

The GCS tracks block access requests within your Real Application Clusters environment, granting requests for resources whenever possible. The GCS also tracks requests for resources that are not currently available. Access rights are granted when these resources later become available. The GCS maintains an inventory of block access requests and status of resources.

Interrupt and Completion Processing

There are three situations where processes are interrupted or notified to handle a request for a data block:

- When a block is received from another cache
- When read and write permissions are granted on a resource and no block was transferred because no other instance had cached it
- When a request is blocked on another instance.

The usual flow of a Cache Fusion request is that a block request is made to the GCS and forwarded to the instance in which the data is cached. From there, the buffer is sent directly to the requestor, which is interrupted and completes the request. One key part of request completion is that the requestor informs the GCS that it has received the block. This is called a *block arrival interrupt*. It also informs the GCS that is taking it in a particular mode and role. This is called the *assume notification*. Informing the GCS is an asynchronous task. (That is, it is not blocking.)

On the holding side, an interrupt occurs when the resource requested by another instance is held in a conflicting mode. When this occurs, processes on the holding instance are interrupted. Processes on the holding instance are interrupted in order to release or downgrade their access privileges and send the block. This is called a blocking notification or **blocking interrupt**.

In some cases the GCS determines that resource is not available in any other instance in the cluster and grants permission to access the block directly. Upon request completion the requesting process will then read the block from disk. If the GCS can make the decision to grant the request locally (that is, without sending messages) the request will be completed immediately. Otherwise, an **acquisition interrupt** is sent to the requesting process.

The **Global Enqueue Service (GES)** uses a similar notification mechanism. There, only completion interrupts and blocking interrupts are used.

All requests for cluster-wide access to a resource are maintained in grant queues and convert queues. While requests are in progress and until they are completed, the requests remain in a convert queue. These queues are managed by the GCS and GES.

Block Access Requests are Queued

The GCS maintains two queues for resource requests:

Convert queue	If the GCS cannot immediately grant a resource request, then the GCS places the request in the convert queue where it tracks waiting resources requests.
Granted queue	The GCS tracks resource requests that have been granted in the granted queue.

Acquisition Interrupts Communicate Block Access Request Status

To communicate the statuses of resource requests, the GCS uses two types of interrupts (also known as **wake up call**s):

Block Access Requests are Granted and Converted

The following figures show how the GCS handles resource requests. In Figure 5–5, shared request 1 has been granted on the resource to process 1, and shared request 2 has been granted to process 2. As mentioned, the GCS tracks the resources in the granted queue. When a request for an exclusive block access mode is made by process 2, it must wait in the convert queue.

Figure 5–5 The Global Cache Service Grants and Converts Queues

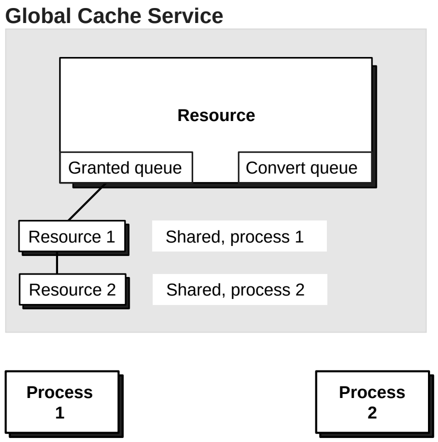
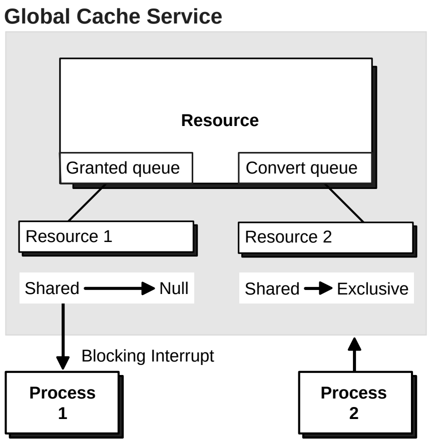


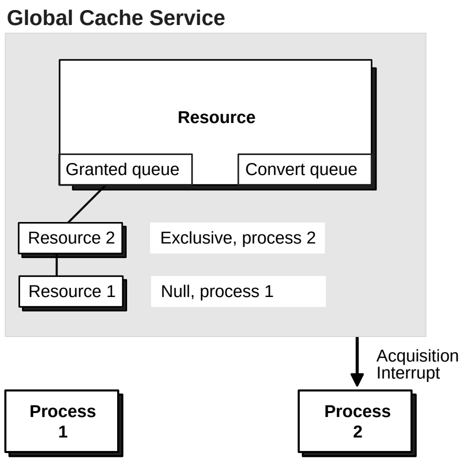
Figure 5–6 shows the GCS sending a **blocking interrupt** to Process 1, the owner of the shared resource, notifying it that a request for an exclusive resource is waiting. When the shared resource is relinquished by Process 1, Oracle converts the access mode NULL or releases it.

Figure 5–6 Blocking Interrupt



An acquisition interrupt is then sent to alert Process 2, the requestor of the exclusive resource. The GCS grants the exclusive resource and converts it to the granted queue. Figure 5–7 illustrates this.

Figure 5–7 Function of an Acquisition Interrupt



Recovery in Real Applications Clusters

Real Application Clusters recovery is optimized to execute certain steps in parallel. Data blocks become available immediately after they are recovered. Database recovery and resource space reconfiguration are divided into two phases and can be executed in parallel. Generally, the recovery process allows a high degree of parallelism and hence better availability and scalability.

When an instance expires and the failure is detected by another Oracle instance in the cluster, Oracle performs the following recovery steps:

1. GCS resources and write requests are frozen while GES enqueues are reconfigured.
2. After the reconfiguration of the enqueues that are controlled by the GES, the following take place in parallel: a log read, recovery, and remastering of GCS resources. At the end of this step, the resources of the blocks that need to be recovered have been identified and the Global Resource Directory is reconstructed. Pending requests or writes have been cancelled or replayed.
3. Buffer space for recovery is allocated and the resources identified in the previous pass over the log are claimed as recovery resources. Then, assuming that there are PIs of blocks to be recovered in other caches in the cluster, source buffers are requested from other instances. The resource buffers are the starting point of recovery for a particular block.

4. All resources and enqueues required for subsequent processing have been acquired and the Global Resource Directory is now *unfrozen*. Any data blocks that are not in recovery can now be accessed. Note that the system is already partially available.
5. The cache layer recovers and writes each block identified in step 2, releasing the recovery resources immediately after block recovery so that more and more blocks become available as cache recovery proceeds.
6. After all blocks have been recovered and recovery resources have been released, the system is again fully available.

In summary, the recovered database or recovered portions of the database become available earlier, and before the completion of the entire recovery sequence. This makes the system more available and recovery more scalable.

In the rare occurrence of multiple simultaneous instance failures, neither the PI buffers nor the current buffer for a data block can be found in any of the surviving instances' caches. Then a log merge of the failed instances must be performed. The performance penalty of a log merge is proportional to the number of failed instances and the size of the redo logs for each instance. The size of the log to be read can be controlled by checkpoint features.

With its advanced design, Real Application Clusters recovery is able to handle multiple simultaneous failures and sequential failures. The shared cache server is also resilient to instance failures or crashes during recovery.

6

Coordination by the Global Enqueue Service

This chapter details the interinstance coordination activities that take place in Real Application Clusters. The **Global Enqueue Service (GES)** coordinates **resource**, data, and interinstance data requests. This chapter describes the details about how Oracle coordinates these resources with enqueues generated by the GES.

This chapter primarily discusses GES enqueues. Global Cache Service (GCS) resources are discussed in Chapter 5, "Cache Fusion and the Global Cache Service".

Topics in this chapter include:

See Also: Chapter 5, "Cache Fusion and the Global Cache Service" for information on the Global Cache Service (GCS)

Cache Synchronization

Understanding how Real Application Clusters synchronize caches across **instance**s can help you understand the overhead affecting system performance. Consider a five-node Real Application Clusters environment where a user drops a table on one of the nodes. Since each of the five **dictionary cache**s has a copy of the definition of the dropped table, the node dropping the table from its cache must also cause the other four dictionary caches to drop their copies of the table. Real Application Clusters handles this automatically through the GES. Users on the other nodes are notified of the change in resource status.

There are significant advantages to having each node store library and table information. Occasionally, the `DROP TABLE` statement forces other caches to be flushed, but the brief effect this has on performance does not necessarily diminish the advantage of having multiple caches.

Each instance of Real Application Clusters has a dictionary cache, or **row cache**, containing data dictionary information in its **System Global Area (SGA)**. The data dictionary structure is the same for Oracle instances in **cluster database**s as for instances in exclusive mode. Real Application Clusters uses global enqueues to coordinate data dictionary activity among multiple instances.

Well-designed applications used in conjunction with Real Application Clusters can achieve significant speed up and scale up.

Global Enqueue Service Concurrency Control

The GES tracks the status of all Oracle enqueuing mechanisms. Oracle only creates global enqueues if you start an Oracle instance with Real Application Clusters enabled. The exception to this is the mount lock. The GES synchronizes global enqueues by communicating the status of an enqueue's status to all instances within a cluster.

An instance *owns* a global **enqueue** that *protects* a resource, such as a data block or data dictionary entry, when the resource enters the SGA of an instance. The GES manages resources accessed by more than one instance.

Concurrency for Database Global Enqueues

Apart from the mechanism to guarantee coherence of cached database blocks, other layers in Oracle must globally protect their data structures. These other layers are, for example, the:

- Data Dictionary Access Layer
- Library Cache Layer
- Transaction Management Code
- Recovery Code

Each layer has specified needs, semantics, and protocols for access to their respective data structures. All layers of Oracle use the service of the GES API to acquire, convert, and release resources.

You do not need to tune global enqueues. In addition, the GES automatically calculates memory requirements when an instance starts up. Oracle uses all parameters affecting global enqueue resources that existed in previous releases to calculate exact resource memory requirements. GES resources include transaction locks, table locks, library cache locks, and dictionary cache locks.

Global enqueue resources are sometimes held only for a very short time and quickly released. Normally, TX locks, for example, are acquired when a transaction starts. They are released when the transaction commits. In some cases they are used to signal an event, such as in the case of library cache locks, where DDL statements may cause invalidations of library cache objects. However, you do not have to tune or monitor these resources.

Aside from the global enqueue resources in the previous list, Oracle databases have a variety of other global enqueue resource types. However, these resources do not have such an important role as those listed. The next section more closely examines the important global enqueue resource types.

Global Enqueue Service Processing

The GES controls access to datafiles and control files. The GES also controls library and dictionary caches, and perform various types of communication between instances. These enqueues do not protect datafile blocks. Examples of GES enqueues are Data Manipulation Language enqueues, transaction locks, and **Data Definition Language (DDL)** locks or dictionary locks. GES processing includes coordination for enqueues other than data blocks.

The resources described in the following section are local in single instance Oracle, but global in the GES.

The resources managed by the GES include:

- Transaction Locks
- Table Locks
- Library Cache Locks
- Dictionary Cache Locks
- Database Mount Lock

> **See Also:** *Oracle9i Real Application Clusters Deployment and Performance* for details on calculating the number of GES processing enqueues to configure in the Global Cache Service

This section explains how Oracle uses GES enqueues to manage concurrency for transactions, tables, and other entities within an Oracle environment.

Transaction Locks

A transaction lock is acquired in exclusive mode when a transaction initiates its first row level change. The lock is held until the transaction is committed or rolled back. The **system monitor (SMON)** also acquires a transaction lock in exclusive mode when recovering (undoing) a transaction. Transaction locks are used as a queuing mechanism for processes awaiting the release of an object locked by a transaction in progress.

Table Locks

Table locks are GES locks that protect entire tables. A transaction acquires a **table lock** when a table is modified by one of the following statement types: `INSERT`s, `UPDATE`s, `DELETE`s, `SELECT`s with the `FOR UPDATE` clause, and `LOCK TABLE`. A table lock can be held in any of several modes: null (N), row share (RS), row exclusive (RX), share lock (S), share row exclusive (SRX), and exclusive (X).

When an instance attempts to mount the database, a table lock is used to ensure that all participating instances either have `DML_LOCKS = 0` or `DML_LOCKS != 0`. If they do not, then Oracle displays error ORA-61 and the mount attempt fails. Table locks are acquired during the execution of a transaction when referencing a table with a DML statement so that the object is not dropped or altered during the execution of the transaction. This occurs if and only if the DML_LOCKS parameter is nonzero.

You can also selectively turn table locks on or off for a particular table, using the statement:

```
ALTER TABLE tablename DISABLE|ENABLE TABLE LOCK
```

If you set DML_LOCKS to zero, then DDL operations are not allowed. The same is true for tables that have disabled table locks.

> **See Also:** *Oracle9i Real Application Clusters Deployment and Performance* for more information about minimizing instance locks and disabling table locks for improved performance

Library Cache Locks

When a database object (such as a table, view, procedure, function, package, package body, trigger, index, cluster, or synonym) is referenced during parsing or compiling of a **structured query language (SQL)**, (Data Manipulation Language (DML) or **Data Definition Language (DDL)**, **PL/SQL**, or Java statement, the process parsing or compiling the statement acquires the library cache lock in the correct mode. In Oracle9*i*, the lock is held only until the parse or compilation completes (for the duration of the parse **call**).

Dictionary Cache Locks

The **Data Dictionary Cache** contains information from the data dictionary, the metadata store. This cache provides efficient access to the data dictionary.

Creating a new table, for example, causes the metadata of that table to be cached in the data dictionary. If a table is dropped, then the metadata needs to be removed from the data dictionary cache. To synchronize access to the data dictionary cache, latches are used in exclusive mode and in single node cluster databases. Global enqueues are used in cluster database mode.

In Real Application Clusters, the data dictionary cache on all nodes can contain the metadata of a table that gets dropped on one instance. The metadata for this table needs to be flushed from the data dictionary cache of every instance. This is performed and synchronized by using global enqueues.

Database Mount Lock

The **Database Mount Lock** shows whether an instance has mounted a particular database. This lock is only used with Real Application Clusters and it is the only multi-instance lock used by the Real Application Clusters software in exclusive mode, where it prevents another instance from mounting the database in shared mode.

In Real Application Clusters single node cluster databases, this lock is held in shared mode. Another instance can successfully mount the same database in shared mode. In exclusive mode, however, another instance is unable to obtain the lock.

See Also: *Oracle9i Database Reference* or more information about transaction locks, table locks, library cache locks, and dictionary cache locks.

Deadlock Detection in the Global Enqueue Service

The GES performs deadlock detection to all deadlock sensitive enqueues and resources. It does *not* control access to tables or objects in the database itself. Real Application Clusters uses the GES to coordinate concurrent access across multiple instances to resources such as data blocks and rollback segments.

Part III

Implementing Real Application Clusters

Part Three describes several topics specific to Real Application Clusters implementation. The chapters in this part are:

7

Real Application Clusters Components

This chapter describes the components used for implementing Real Application Clusters. The topics in this chapter are:

- Instance and Database Components for Real Application Clusters
- System Change Number Processing

Instance and Database Components for Real Application Clusters

This chapter discusses the concepts of implementation specific to Real Application Clusters from a process and component point-of-view. Each cluster database instance has its own **System Global Area (SGA)** and background processes common to single instances. Instances within a cluster also share or have access to all SGAs, control and datafiles, and redo logs throughout the cluster.

Other components specific to Real Application Clusters are described in the following sections. These components enable cluster database processing and facilitate internode communication.

Real Application Clusters Processes

In addition to the processes common to single instances, such as the **process monitor (PMON)**, **system monitor (SMON)**, and **database writer (DBWR)**, Real Application Clusters instances have additional processes to facilitate resource sharing among nodes in a cluster.

> **See Also:** *Oracle9i Database Concepts* for more information on Oracle processes

Several processes used by Real Application Clusters facilitate resource sharing. These work with the Global Cache Service component to manage global enqueues and resources:

- The **Global Enqueue Service Monitor (LMON)** monitors the entire cluster to manage global enqueues and resources. LMON manages instance and process expirations and the associated recovery for the Global Cache Service. In particular, LMON handles the part of recovery associated with cluster enqueues.
- The **Global Enqueue Service Daemon (LMD)** is the lock **agent** process that manages enqueue manager service requests for Global Cache Service enqueues to control access to global enqueues and resources. The LMD process also handles deadlock detection and remote enqueue requests. Remote enqueue requests are requests originating from another instance.
- The **Global Cache Service Processes (LMSn)** are processes for the **Global Cache Service (GCS)**. Real Application Clusters software provides for up to 10 Global Cache Service processes. The number of LMSn varies depending on the amount of messaging traffic among nodes in the cluster. The LMS*n* handle **blocking interrupt**s from the remote instance for Global Cache Service

resources. For cross-instance read / write requests, the LMS*n* creates consistent read versions of blocks and sends blocks to the requesting instance. The LMS*n* also control the flow of messages to remote instances.

- The **LCK process** manages global enqueue requests and cross-instance broadcast.

When an instance starts, the LMON and LMD processes start and register with the Cluster Manager. The LMON de-registers with the Cluster Manager when you shut down the database.

When an instance fails while in shared mode, another instance's SMON detects the failure and recovers the failed instance. The LMON process of the instance performing recovery remasters outstanding Global Cache Service resources and **Global Enqueue Service (GES)** enqueues for the failed instance.

Foreground Global Cache Lock Element Acquisition

Foreground processes are the processes associated with user sessions. One instance communicates enqueue requests directly to remote LMD processes in other instances. Foreground processes request information such as the resource names of the enqueues they are requesting, and their requisite enqueue modes. The Global Cache Service processes the request asynchronously. The foreground process therefore waits for request completion before closing the request.

Cache Fusion Processing

To process a Cache Fusion request, a sequence of steps must be completed. Depending on the state of the enqueues and resources, multiple processes are involved. If one or more **Global Cache Service Processes (LMSn)** are active, then the steps are as follows:

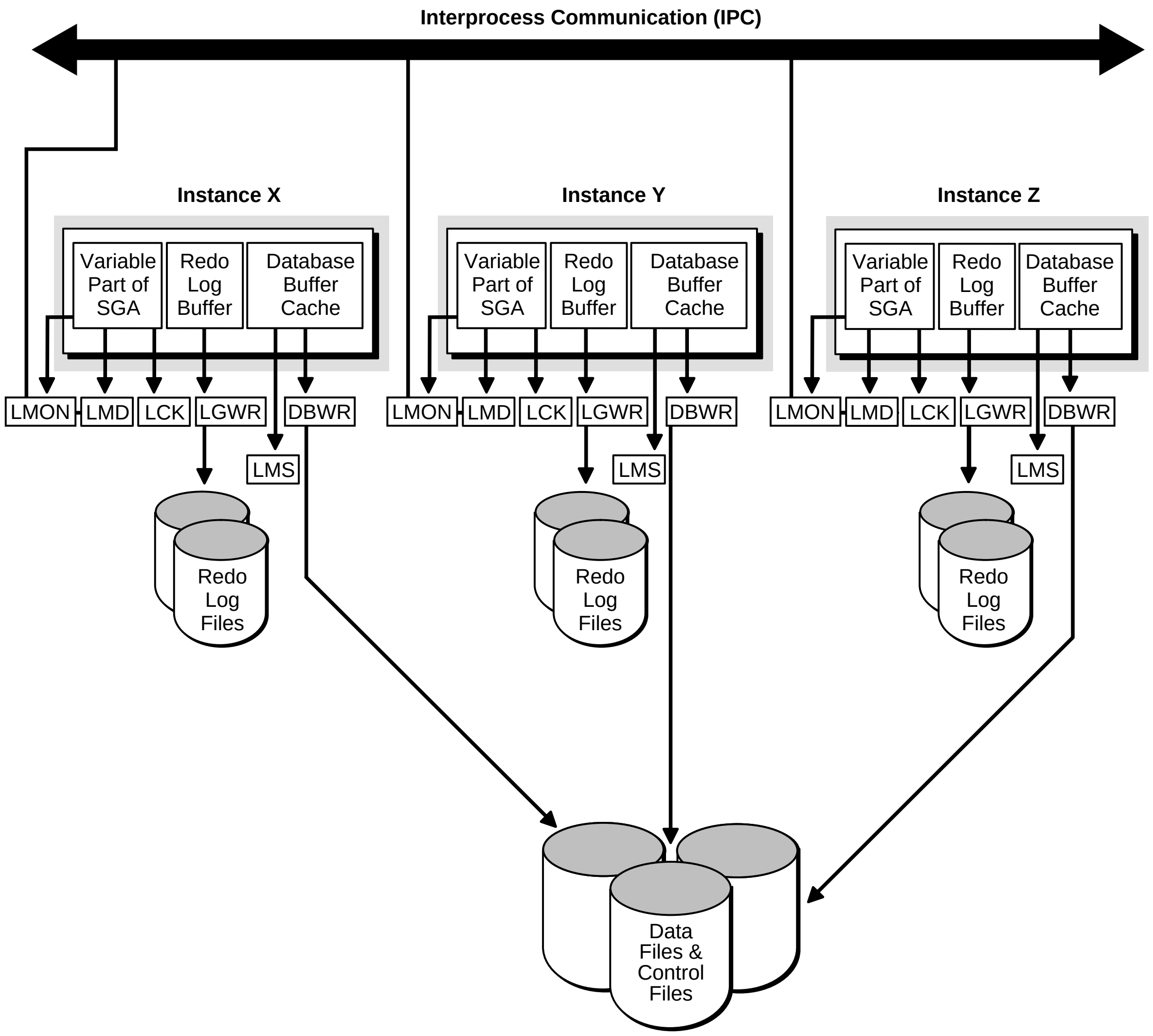
- The LMD receives all messages and dispatches most Global Cache Service messages to the LMS*n* for processing. The Global Cache Service tracks the state of most requests as well as the location and version of the latest **past image (PI)** block. If a resource request is granted by the master instance, the LMS*n* executes the **acquisition interrupt** handler and posts the requesting foreground process. The LMS*n* also normally executes the **blocking interrupt** routine and perform the down convert to satisfy the resource request from a remote instance.
- The LMS*n* sends current blocks to another instance. It also processes *down converts* when a block is requested by another instance and notifies the log writer to write redo or block-written records to disk. When processing a

consistent read request, the LMS*n* sends data blocks directly to the requesting instance.

Overview of Real Application Clusters Processes

The processes specific to Real Application Clusters and some of the processes that Oracle also uses in single instance environments appear in Figure 7–1.

Figure 7–1 Basic Elements of Real Application Clusters



System Change Number Processing

The **System Change Number (SCN)** is a logical time stamp that Oracle uses to order events within a single instance and across all instances. The main purpose of SCNs is to mark redo log entries to synchronize recovery processing. Oracle assigns an SCN to each transaction. Conceptually, there is a global serial point that generates SCNs. In practice, however, SCNs can be read and generated in parallel. One of the SCN generation schemes is called Lamport SCN generation.

In single instance Oracle, the **System Global Area (SGA)** maintains and increments SCNs from an instance that has mounted the database in exclusive mode. In Real Application Clusters shared mode, the SCN must be maintained globally. Its implementation can vary from platform to platform. The SCN can be handled by the Global Cache Service, by the Lamport SCN generation scheme, or by using a hardware clock or dedicated SCN server.

Lamport SCN Generation

The Lamport SCN generation scheme is fast and scalable because it generates SCNs in parallel on all instances. In this scheme, all messages between instances carry SCNs. Multiple instances can generate SCNs in parallel, with no need for extra communication among these instances.

On most platforms, Oracle uses the Lamport SCN generation scheme when the `MAX_COMMIT_PROPAGATION_DELAY` is larger than a platform-specific threshold. This is generally the default. This value is typically set to 7 seconds. Note that by changing this value you change the SCN generator used by the database. You can examine the alert log after instance startup to see whether the Lamport SCN generation scheme is in use. If this value is smaller than the threshold value, then Oracle uses the lock SCN generation scheme.

8 Real Application Clusters Storage Considerations

This chapter describes storage considerations for Real Application Clusters applications. Topics in this chapter include:

- Storage Issues
- Space Management and Free List Groups
- Free List Group Examples
- Controlling Extent Allocation

Storage Issues

This section describes storage issues specific to Real Application Clusters. This section discusses:

- Datafile Storage in Real Application Clusters
- Parameter File Storage in Real Application Clusters
- Redo Log File Storage in Real Application Clusters
- Rollback Segments in Real Application Clusters

Datafile Storage in Real Application Clusters

All Real Application Clusters instances access the same datafiles. The composition of database files is the same for Oracle in parallel mode and exclusive mode. You do not have to alter the datafiles to start Oracle in parallel mode.

To improve performance, you can control the physical placement of data so that the instances use separate sets of data blocks. Free list groups, for example, enable you to allocate space for inserts to particular instances.

Whenever an instance starts up, it verifies access to all online datafiles. The first Real Application Clusters instance to start must verify access to all online files so it can determine if media recovery is required. Additional instances can operate without access to all of the online datafiles, but any attempt to use an unverified file fails and an error message is generated.

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

See Also: *Oracle9i Recovery Manager User's Guide* for a description of Block Media Recovery (BMR).

When an instance adds a datafile or brings a datafile online, all instances verify access to the file. If an instance adds a new datafile on a disk that other instances cannot access, verification fails, but the instances continue running. Verification can also fail if instances access different copies of the same datafile.

If verification fails for any instance, then diagnose and fix the problem, and use the `ALTER SYSTEM CHECK DATAFILES` statement to verify access.

See Also:

- *Oracle9i SQL Reference* for details on using the `ALTER SYSTEM CHECK DATAFILES` statement
- *Oracle9i Database Administrator's Guide* for information on the Oracle Managed File (OMF) feature and related file naming conventions
- *Oracle9i Real Application Clusters Installation and Configuration* for special naming conventions for datafiles as specified in Real Application Clusters

Parameter File Storage in Real Application Clusters

Real Application Clusters records parameter settings for your database in a server initialization text file that resides on the server. The **Database Configuration Assistant (DBCA)** creates this file as a binary server parameter file when a database is created. The server parameter file records values for both global and instance-specific parameter settings.

Oracle uses parameter settings in parameter files to determine how to control various database resources. You can use two types of files for parameter administration: the server-side parameter file or one or more traditional client-side parameter files.

Oracle Corporation recommends that you administer parameters using server parameter files. If you use the traditional client-side parameter files, parameter changes that Oracle makes as a result of self-tuning are not preserved after shutdown.

See Also: *Oracle9i Real Application Clusters Administration* for more information on using client-side parameter files

Using Server Parameter Files in Real Application Clusters

By default, the Oracle creates server parameter files based on one **parameter file (PFILE)**. You can only change parameter settings in server parameter files using **Oracle Enterprise Manager** or `ALTER SYSTEM SET SQL` statements. These are binary files that you cannot edit directly.

Note: Oracle Corporation recommends that you avoid modifying the values for self-tuning parameters; overriding these settings can adversely affect performance.

Location of Server Parameter Files The default location of the server parameter file is inappropriate for Real Application Clusters. This is because all instances must use the same server parameter file.

The server parameter file uses the Oracle **system identifier (SID)** designator to indicate global parameter settings. Although you can still use client-side parameter files, Oracle strongly recommends that you use server parameter files in Real Application Clusters environments. Note that server parameter files are also used with single instance databases.

Advantages of Using Server Parameter Files By using server parameter files, you can take advantage of Oracle's advanced self-tuning features. When Oracle self-tunes, it can only do so by automatically modifying parameter settings in server parameter files. Using server parameter files also greatly simplifies parameter administration within Real Application Clusters environments.

See Also: *Oracle9i Real Application Clusters Installation and Configuration* for details on server parameter files

Redo Log File Storage in Real Application Clusters

In Real Application Clusters, each instance writes to its own set of online redo log files. The redo written by a single instance is called a **thread** of redo. Each online redo log file is associated with a particular thread number. When an online redo log is archived, Oracle records its thread number to identify it during recovery.

A *private thread* is a redo log created by using the `ALTER DATABASE ADD LOGFILE` statement with the `THREAD` clause. A *public thread* is a redo log created using the `ALTER DATABASE ADD LOGFILE` statement but without specifying a `THREAD` clause.

If the `THREAD` initialization parameter is specified, then the instance starting up acquires the thread identified by that value as a *private thread*. If `THREAD` has the default value `0`, then the instance acquires a public thread. Once acquired, the acquiring instance uses the redo thread exclusively.

Online redo log files can be multiplexed, or *mirrored*.

See Also: *Oracle9i Database Concepts* and *Oracle9i Database Administrator's Guide* descriptions of multiplexed redo log files

Rollback Segments in Real Application Clusters

Rollback segments are records of old values of data that were changed by each transaction (whether or not committed). Oracle requires **rollback segments** to maintain read consistency, to undo changes made by transactions that roll back or abort, and to recover the database.

Each instance must have at least two rollback segments. For private rollback segments, you must always specify the SID when using the `ROLLBACK_SEGMENTS` parameter.

Note: If using automatic undo management, each instance needs its own undo tablespace.

Each instance in Real Application Clusters shares use of the `SYSTEM` rollback segment and requires at least two dedicated rollback segments for each instance. You do not have to manually create an undo **tablespace** or rollback segment unless you manually create your database.

If you use the recommended automatic undo management instead of Rollback Segment Undo, you do not need to do anything more than monitor the use of rollback segments. It is highly recommended that you use automatic undo management instead of Rollback Segment Undo.

See Also: Oracle9i Real Application Clusters Installation and Configuration for greater detail on the use of automatic undo management and Rollback Segment Undo

Both private and public rollback segments can be acquired at instance startup and used exclusively by the acquiring instance. They are used until taken offline or when the acquiring instance is shut down as specified in the rollback segment parameter.

Private rollback segments are unique to a particular instance; other instances cannot use them. A public rollback segment is offline and not used by any instance until an instance that needs an extra rollback segment starts up, acquires it, and brings it online. Once online, the acquiring instance uses the public rollback segment exclusively.

Only one instance writes to a given rollback segment (except for the `SYSTEM` rollback segment). However, other instances can read from it to create read-consistent snapshots or to perform instance recovery.

> **See Also:** *Oracle9i Database Administrator's Guide* for information about contention for a rollback segment and for information on the performance implications of adding rollback segments. This manual also contains details on automatic undo management, undo tablespaces, and managing rollback segments

Space Management and Free List Groups

This section explains space management and **free list group** concepts by covering the following topics:

- How Oracle Handles Free Space
- Free Lists and Free List Groups

How Oracle Handles Free Space

Real Application Clusters enables transactions running on separate instances to insert and update data in the same table concurrently. This occurs without contention to locate free space for new records. However, to take advantage of this capability, you must accurately manage free space in your database by using structures such as free list groups, which are described later in this chapter.

Oracle keeps track of blocks with space available for transactions that could cause rows to exceed the space available in their original block. Oracle does this for each database object, such as a table, cluster, or index. A transaction requiring free space can examine the free list of blocks. If the free list does not contain a block with enough space to accommodate it, then Oracle allocates a new extent.

When Oracle allocates new extents to a table, Oracle adds the new extent's blocks to the **master free list**. (The master free list is a list of blocks containing available space drawn from any extent in a table). This can eventually result in contention for free space among multiple instances on Real Application Clusters because the free space contained in newly allocated extents cannot be reallocated to any group of free lists. You can have more control over free space if you specifically allocate extents to instances; in this way you minimize free space contention.

> **See Also:** *Oracle9i Database Concepts* for more information about segments, extents, and the **high water mark**

Free Lists and Free List Groups

Single instance Oracle uses multiple free lists to reduce block contention. Every tablespace has a free list that identifies data blocks with free space. Oracle uses blocks with free space when inserts or updates are made to a database object such as a table.

Blocks in free lists contain free space greater than shown with the PCTFREE parameter. The **PCTFREE** is the percentage of a block reserved for updates to existing rows. In general, blocks included in process free lists for a database object must satisfy the PCTFREE and PCTUSED (percentage used) constraints.

You can specify the number of free lists by setting the FREELISTS parameter when you create a table, index or cluster. The maximum value of the FREELISTS parameter depends on the Oracle block size on your system. In addition, for each free list you need to store a certain number of bytes in a block to accommodate overhead.

Within free list groups there are two subsets of free lists:

- The **master free list**s–lists of blocks containing available space drawn from any extent in a table
- The **transaction free list**s–lists of blocks freed by uncommitted transactions

Because Real Application Clusters has multiple instances, free lists alone cannot solve contention problems. Free list groups, however, effectively reduce forced writes (pinging) between instances.

Note: The reserved area and the number of bytes required for each free list depend upon your platform. For more information, see your Oracle operating system-specific documentation.

Free List Groups

A **free list group** is a set of free lists for use by one or more instances. Each free list group provides free data blocks to accommodate inserts or updates on tables and clusters and is associated with one or more instances at startup. By default, only one free list group is available. This means all free lists for an object reside in the segment header block. Free list groups are supported on *all* database objects.

If multiple free lists reside in a single block in a Real Application Clusters environment, then the block with the free lists could thus experience block writes, or a forced read / write among all the instances. Avoid this by grouping the free lists into separate groups and assigning each group to an instance. Each instance then has its own block containing free lists. Since each instance uses its own free lists, there is no contention among instances to access the same block containing free lists.

Interinstance contention occurs when different instances' transactions insert data into the same table. This occurs because all free lists are held in the segment header if you do not define free list groups. The free list may be from a common pool of blocks, or you can partition multiple free lists so specific extents in files are allocated to objects.

Note: To manage free space efficiently in Real Application Clusters, always use free list groups and free lists.

Avoiding Contention for Segment Headers and Free Lists

A highly concurrent environment has potential contention for the segment header, that contains the free list.

- If free list groups *do* exist, then the segment header only points to the master free list. In addition, every free list group block contains pointers to its own free list.
- If free list groups *do not* exist, then the segment header contains pointers to the free list.

In multiinstance environments, as illustrated in Figure 8–1, free lists provide free data blocks from available extents to different instances. You can partition multiple free lists so that extents are allocated to specific database instances. Each instance hashes to one or more free list groups, and each group's header block points to free lists. Without free list groups, every instance must read the segment header block to access the free lists.

Figure 8–1 Contention for the Segment Header

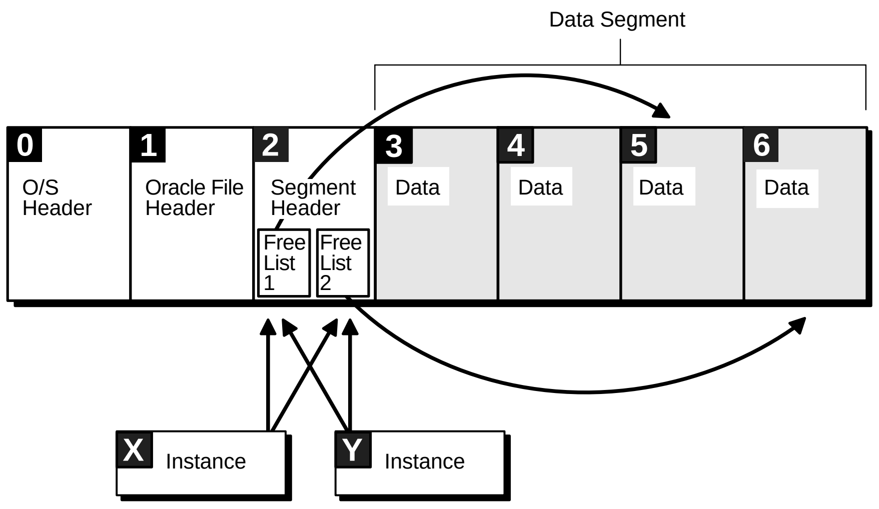
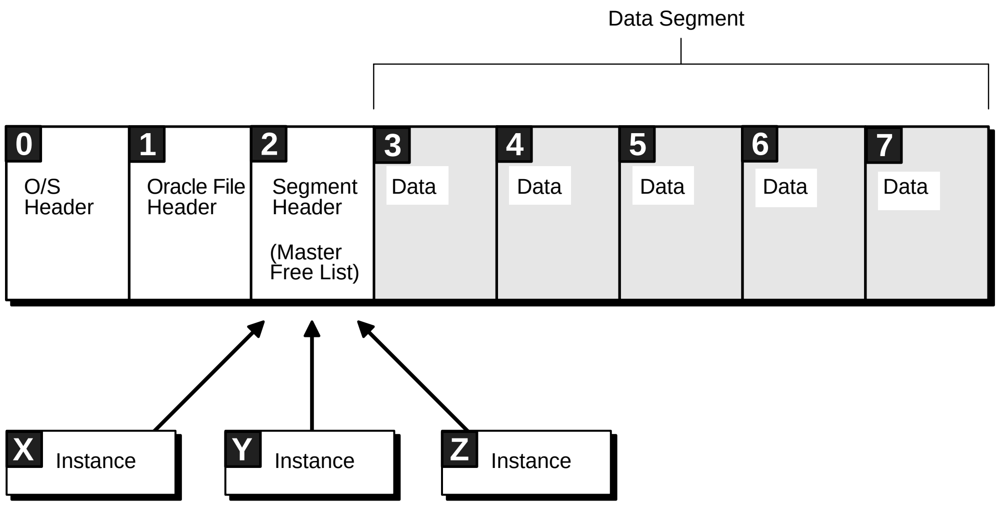


Figure 8–2 shows the blocks of a file where the master free list is stored in the segment header block. Three instances are forced to read this block to obtain free space. Because there is only one free list, there is only one insertion point. Free list groups help reduce contention by spreading this insertion point over multiple blocks. With free list groups each block is accessed less frequently.

Figure 8–2 Contention for the Master Free List



Locally Managed Tablespaces

Locally managed tablespaces are also useful because they help avoid data dictionary contention. This can occur if Real Application Clusters performs a lot of space management for sorting segments in tablespaces, creating or dropping tables, and so on. Locally managed tablespaces eliminate this contention and the benefits of Real Application Clusters in this case easily outweigh the benefits of single instance Oracle.

Free List Group Examples

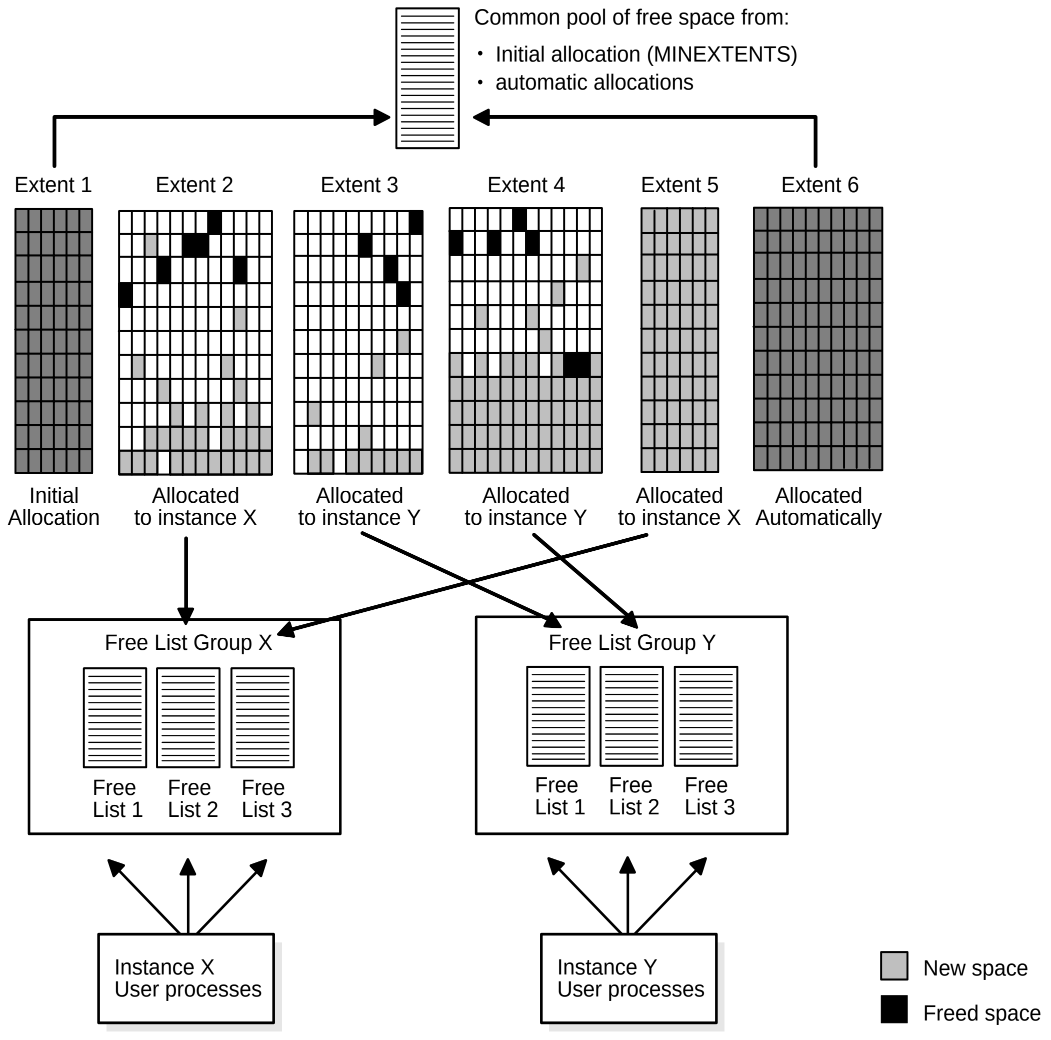
This section describes two free list group examples:

- Basic Free List Group Example
- Complex Free List Group Example

Basic Free List Group Example

Figure 8–3 illustrates the division of free space for a table into a master free list and two free list groups. Each contains three free lists. This example involves a well-partitioned application where deletes occur. The master free list pictured is the master free list for this particular free list group.

Figure 8–3 Groups of Free Lists for a Table



The table shown in Figure 8–3 shows one initial extent. After this, extents 2 and 5 are allocated to instance X, extents 3 and 4 are allocated to instance Y. Extent 6 is allocated automatically, but not to any particular instance. Notice the following:

- The dark gray shaded blocks in the initial allocation in extents 1 and extent 6 represent the master free list of free blocks.
- The light gray blocks in extents 2 and 5 represent available free space in free list group X.
- The light gray blocks in extents 3 and 4 represent the available free space in free list group Y.
- Extent 5 is newly allocated, thus all of its blocks are in free list group X.
- The black blocks represent blocks freed by deletions, that return to free list groups X and Y.
- Unshaded blocks do not contain enough free space for inserts.

Each user process running on instance X uses one of the free lists in group X. Meanwhile, each user process on instance Y uses one of the free lists in group Y. If more instances start up, their user processes share free lists with instance X or Y.

Complex Free List Group Example

The simple case in Figure 8–3 becomes more complicated when you consider that extents are not permanently allocated to instances, and that space allocated to one instance cannot be used by another instance. Each free list group has its own master free list. After allocation, some blocks go onto the master free list for the group, some go to a process free list, and some do not belong to a free list at all. If the application is fully partitioned, then once blocks are allocated to a given instance, they stay with that instance. However, blocks can move from one instance to another if the application is not fully partitioned.

Consider a situation where instance Y fills a block, takes it off the free list, and then instance X frees the block. The block then goes to the free list of instance X, the instance that freed it. If instance Y needs space, then it cannot reclaim this block. Instance Y can only obtain free space from its own free list group

See Also: *Oracle9i Real Application Clusters Installation and Configuration* for details on partitioning data and associating instances, users, and enqueues with free list groups

Controlling Extent Allocation

When a row is inserted into a table and new extents need to be allocated, a certain number of contiguous blocks are automatically allocated by Cache Fusion to the free list group associated with an instance. This **extent allocation** occurs when the table or cluster is first created and new extents that are automatically allocated. These extents add their blocks to the master free list or to the space above the high water mark.

Automatic Allocation of New Extents

When you explicitly allocate an extent without specifying an instance, or when an extent is automatically allocated to a segment because the system is running out of space (the high water mark cannot be advanced any more), the new extent becomes part of the unused space. It is placed at the end of the extent map. This means that the current high water mark is now in an extent *to the left* of the new one. The new extent is thus added *above* the high water mark.

See Also: *Oracle9i Database Concepts* for greater detail on preallocation of new extents, dynamic allocation of blocks on lock boundaries, and on moving a segment's high water mark

9 Scalability in Real Application Clusters

This chapter describes the scalability features of Real Application Clusters. Topics in this chapter include:

Scalability Features of Real Application Clusters

You can implement Real Application Clusters by using several features that enhance application performance and maintain high availability levels. These features:

- Provide persistent connections between clients and your Real Application Clusters database despite failures
- Balance workloads among the nodes by controlling multiple server connections during heavy use periods
- Offer several customizable client connection options to provide seamless failover of application-to-database connections in the event of common types of service disruptions

The **Database Configuration Assistant (DBCA)** automatically configures most of these features for you.

See Also: *Oracle Net Services Administrator's Guide* for information for manually configuring scalability features

Enhanced Throughput: Scale Up

If tasks can run independently of one another, then Real Application Clusters can distribute them to different nodes. This permits you to achieve **scale up**. In essence, scale up is more processes run through the database in the same amount of time. The number of nodes used, however, depends upon the purpose of the system.

If processes can run faster, then the system accomplishes more work. The parallel execution feature, for example, permits scale up. With parallel execution, a system might maintain the same response time if the data queried increased tenfold, or if more users could be served. Real Application Clusters without the parallel execution feature also provides scale up, but does this by running the same query sequentially on different nodes.

With a mixed workload of **Decision Support System (DSS)**, **Online Transaction Processing (OLTP)**, and reporting applications, you can achieve scale up by running multiple programs on different nodes. You can also achieve speed up by rewriting the batch programs and separating their transactions into a number of parallel streams to take advantage of multiple CPUs.

Real Application Clusters also offers improved flexibility by overcoming memory limitations so you can use it to accommodate thousand of users. You can allocate or deallocate instances as necessary. For example, as database demand increases, you can temporarily allocate more instances. Then you can deallocate the instances and use them for other purposes once they are no longer required. This feature is useful in Internet-based computing environments.

Speed-Up and Scale Up: The Goals of Cluster Database Processing

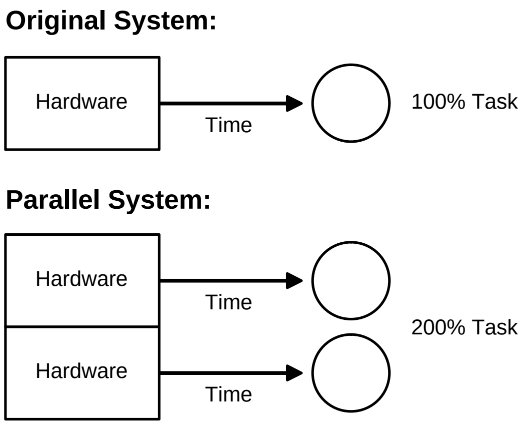
You can measure the performance goals of cluster database processing in two ways:

- Scale Up
- Speed Up

Scale Up

The concept of **scale up** is the factor that expresses how much more work can be done in the same time period by a larger system. With added hardware, a formula for scale up holds the time constant, and measures the increased size of the job that can be done.

Figure 9–1 Scale Up



If transaction volumes grow and you have good scale up, you can keep response time constant by adding hardware resources such as CPUs.

You can measure scale up using this formula:

$$\textit{Scale-up} = \frac{\textit{Volume\_Parallel}}{\textit{Volume\_Original}}$$

Where:

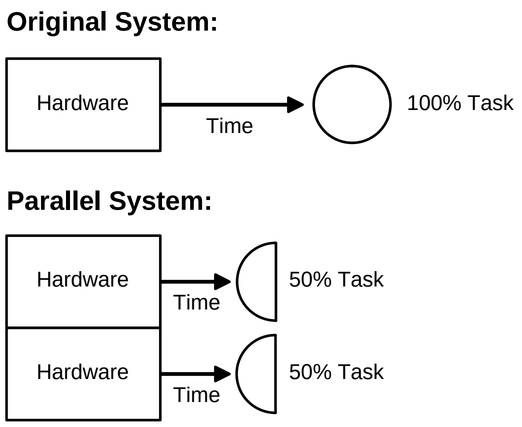
`Volume_Original`	is the transaction volume processed in a given amount of time on a small system.
`Volume_Parallel`	is the transaction volume processed in a given amount of time on a parallel system.

For example, if the original system processes 100 transactions in a given amount of time and the parallel system processes 200 transactions in this amount of time, then the value of scale up would be equal to 2. A value of 2 indicates the ideal of linear scale up: when twice as much hardware can process twice the data volume in the same amount of time.

Speed Up

The concept of **speed up** is the extent that more hardware can perform the same task in less time than the original system. With added hardware, speed up holds the task constant and measures time savings. Figure 9–2 shows how each parallel hardware system performs half of the original task in half the time required to perform it on a single system.

Figure 9–2 Speed Up



Speed up can be calculated with the formula:

Speed Up = Time1 volume / volume - Time2 volume / volume

(Where the volume of work is the same *n* transactions.)

Note that you might not experience direct, linear speed up. Instead, speed up could be more logarithmic. That is, assume the system can perform a task of size *x* in a time duration of *n*. But for a task of size 2*x*, the system might require a time duration of 3*n*.

Where are Cluster Databases Advantageous?

This section describes applications that benefit from Real Application Clusters. It includes the following topics:

- Transaction Systems
- Decision Support Systems

Transaction Systems

The applications in the **transaction systems** category include e-business as well as more traditional online transaction processing (OLTP) systems. Transaction systems are characterized by the use of relatively short transactions that update the database with the capture of specific information such as Internet purchases, credit card purchases, or telephone order purchases. In general, the transaction must be completed within a short response time. The greatest challenge is to scale up transaction systems to support many concurrent transactions.

Transaction systems benefit from Real Application Clusters by scale up. Cache Fusion enables updates to occur from multiple instances with minimal overhead, due to concurrency control. Well-designed applications that scale well on a single system with a single instance of Oracle9*i* will be able to scale in a clustered configuration with Real Application Clusters. As transaction systems are required to handle more transactions as more users are added to the system, additional capacity can be added by simply adding nodes to the cluster. However, poorly designed systems will not scale well and may experience a degradation of performance.

Decision Support Systems

Data warehousing applications that infrequently update, insert, or delete data are often appropriate for using the Real Application Clusters. Query-intensive applications and other applications with low update activity can access the database through different instances with little additional overhead.

If the data blocks are not modified, then multiple nodes can read the same block into their buffer caches and perform queries on the block without additional I/O or enqueue operations.

Decision support applications are good candidates for using Real Application Clusters because they only occasionally modify data, as in a database of financial transactions that is mostly accessed by queries during the day and is updated during off-peak hours. Decision support systems usually benefit from scale up and may experience speed up as well.

Oracle Parallel Execution on Real Application Clusters

Real Application Clusters provides the framework for parallel execution to operate between nodes. **Oracle parallel execution** behaves the same way with or without Real Application Clusters. The only difference is that Real Application Clusters enables parallel execution to distribute portions of statements among nodes so that the nodes execute on behalf of a single query. The server sub-divides the statement into smaller operations that run against a common database residing on shared disks. Because parallel execution is performed by the server, this parallelism can occur at a low level of server operation.

If Oracle does not process a statement in parallel, then Oracle reads disks serially with one I/O stream. In this case, a single CPU scans all rows in a table. With the statement parallelized, disks are read in parallel with multiple I/Os.

Several CPUs can each scan a part of the table in parallel and aggregate the results. Parallel execution benefits not only from multiple CPUs but also from greater I/O bandwidth availability.

Oracle parallel execution can run with or without Real Application Clusters. Without Real Application Clusters parallel execution cannot perform multinode parallelism. Real Application Clusters optimizes the Oracle9*i* Enterprise Edition running on clustered hardware using a parallel cache architecture to avoid shared memory bottlenecks in OLTP and DSS applications.

Levels of Scalability

Successful implementation of cluster database processing and parallel databases requires optimal **scalability** on these levels:

- Network Scalability
- Operating System Scalability
- Database Management System Scalability

Note: Inappropriately designed applications might not fully utilize the potential scalability of the system. Likewise, no matter how well your applications scale, you will not get the desired performance if you try to run them on hardware that does not scale.

Network Scalability

Interconnects are key to hardware scalability. That is, every system must have some means of connecting the CPUs, whether this is a high speed bus or a low speed Ethernet connection. Bandwidth and latency of the interconnect then determine the scalability of the hardware.

Network Scalability as a Function of Bandwidth and Latency

Most interconnects have sufficient bandwidth. A high bandwidth might, in fact, disguise high latency.

Hardware scalability depends heavily on very low latency. Resource coordination traffic is characterized by a large number of very small messages in the LMD process.

Consider an example of a highway and the difference between conveying a hundred passengers on a single bus, compared to one hundred individual cars. In the latter case, efficiency depends largely upon the ability of cars to quickly enter and exit the highway. Even if the highway has 5 lanes so multiple cars can pass, if there is only a one-lane entrance ramp, there can be a bottleneck getting onto the *fast* highway.

Other operations between nodes, such as parallel execution, rely on high bandwidth.

Network Scalability Overview of Client/Server Connectivity

In Real Application Clusters, client/server connections are established by using several subcomponents. A client submits a connection request to a listener process that resides on the destination node in the cluster. A **listener** process is a process that manages incoming connection requests. The listener grants a client/server connection by way of a particular node based on several factors depending on how you configure the connection options.

Enhanced Network Scalability by Using the Shared Server

The shared server enables enhanced scalability in that for a given amount of memory it enables you to support a larger number of users than when you use dedicated servers. The incremental costs of memory as you add a user is less than when you use a dedicated server. In addition to improving the scalability of the number of users, shared server offers additional benefits.

An increasing number of Oracle9*i* features require the shared server. You need the shared server if you use the following database features:

- Oracle9*i* JServer
- Connection Pooling
- Connection Concentration Feature
- Advanced Queueing

See Also: *Oracle Net Services Administrator's Guide* for shared server configuration information for Real Application Clusters environments

Service Registration and the Shared Server Functionality

An important feature of the shared server process is **service registration**. This feature provides the listener with the **service name**s, **instance name**s and network addresses of the database, as well as the current state and load of all instances and shared server dispatchers. With this information, the listener can forward client connection requests to the appropriate dispatchers and dedicated servers.

Because this information is registered with the listener, you do not need to configure the **listener.ora** file with this static information about a database service. Service registration also extends benefits to connection load balancing that is a feature of the shared server.

Note: Service registration is also available in dedicated servers. However, by their nature dedicated servers do not provide the same potential scalability as shared servers.

Connection Load Balancing

The connection load balancing feature provides exceptional benefits in Real Application Clusters environments where there are multiple instances and dispatchers. Connection load balancing improves performance by balancing the number of active connections among various instances and shared server dispatchers for the same service. Note that if your application is partitioned that you may not want to use this feature.

Because of service registration’s ability to register with remote listeners, a listener is always aware of all instances and dispatchers regardless of their location. This way, a listener can send an incoming client request for a specific service to the least-loaded instance and least-loaded dispatcher.

Service Registration also facilitates connect-time failover and client load balancing that you can use with or without a shared server.

Connect-Time Failover for Multiple Listeners

Service registration enables listeners to know whether an instance is up at all times. This enables you to use connect-time failover when multiple listeners support a service. Do this by configuring a client to failover client requests to a different listener if the first listener connection fails. The reconnection attempts continue until the client successfully connects to a listener. If an instance is down, then a listener returns a network error.

Client Load Balancing for Multiple Listeners

When more than one listener supports a service, a client can randomly send connection requests to the various listeners. The random nature of the connection requests distributes the load to avoid overburdening a single listener.

In addition to load balancing, clients can also specify that their connection request automatically fails over to a different listener if a connection cannot be made with the listener chosen at random. A connection could fail either because the specified listener is not up or because the instance is not up and the listener therefore cannot accept the connection.

See Also: *Oracle Net Services Administrator's Guide* for configuration information on the features described in this section

Operating System Scalability

The ultimate scalability of your system also depends upon the scalability of the operating system. This section explains how to analyze this factor.

Software scalability can be an important issue if one node is a shared memory system (that is, a system where multiple CPUs connect to a symmetric multiprocessor single memory). Methods of synchronization in the operating system can determine the scalability of the system. In asymmetrical multiprocessing, for example, only a single CPU can handle I/O interrupts. Consider a system where multiple user processes request resources from the operating system.

Database Management System Scalability

An important distinction in Real Application Clusters architecture is internal versus external parallelism; this has a strong effect on scalability. The key difference is whether the **Object-Relational Database Management System (ORDBMS)** parallelizes the query, or an external process parallelizes the query.

The **resource affinity** capability of Real Application Clusters can improve performance by ensuring that nodes mainly access local, rather than remote, devices. An efficient synchronization mechanism enables better speed up and scale up.

See Also:

- *Oracle9i Real Application Clusters Deployment and Performance*
- *Oracle9i Database Performance Guide and Reference*

Application Scalability

Application design is key to taking advantage of the scalability of the other elements of the system.

No matter how scalable the hardware, software, and database might be, a table with only one row that every node is updating will synchronize on one data block. It is best to use sequences to improve scalability, for example by using the SQL command `INSERT INTO ORDERS VALUES`.

You can preallocate and cache sequence numbers to improve scalability. However you might not be able to scale some applications due to business rules. In such cases, you must determine the cost of the rule.

Note: Clients must be connected to server machines in a scalable manner. This means that your network must also be scalable!

See Also: *Oracle9i Real Application Clusters Deployment and Performance* for information on designing databases and application analysis, and on the sequence number generator

Part IV

High Availability

Part Four includes the following information on Oracle's products and features that provide high availability:

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**.

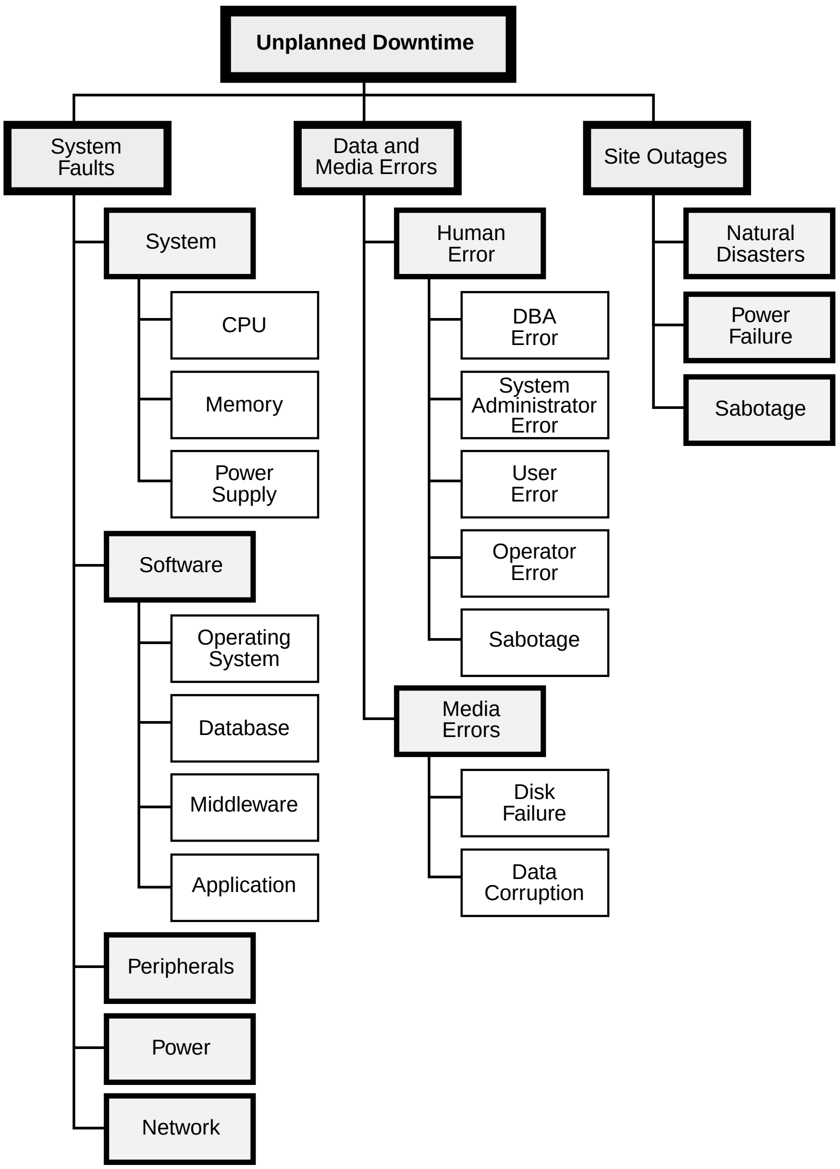
It is difficult to consider several nines of availability without also describing the ground rules and strict processes for managing application environments, testing methodologies, and change management procedures. For these reasons, Real Application Clusters can significantly reduce MTTR during failures.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

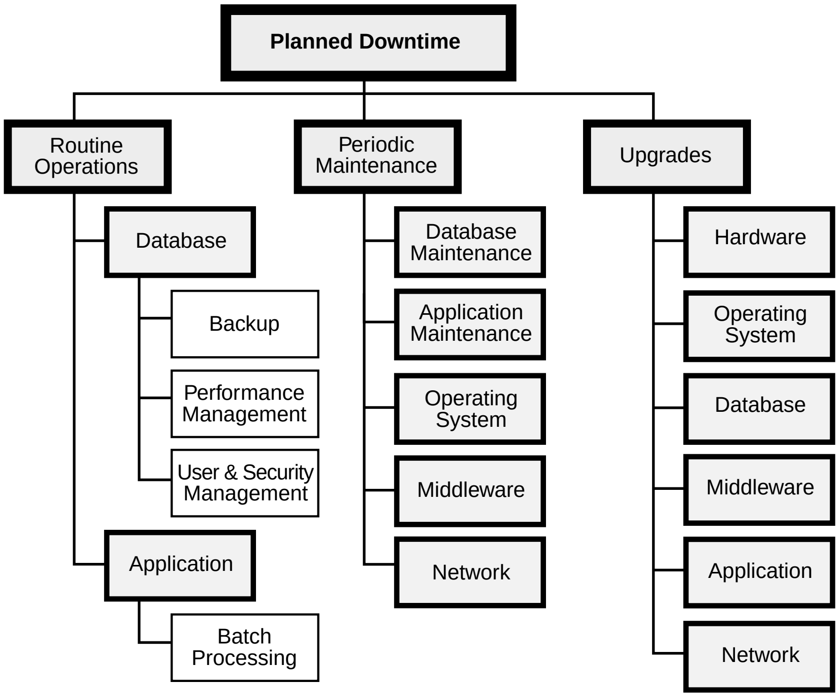
Downtime can be either planned or unplanned. Figure 10–1 shows many causes of **unplanned downtime**. They can be broadly classified as system faults, data and media errors, and site outages. This unplanned downtime is disruptive because it is difficult to predict its timing.

Figure 10–1 Causes of Unplanned Downtime



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



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: *Oracle 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:** *Oracle 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:

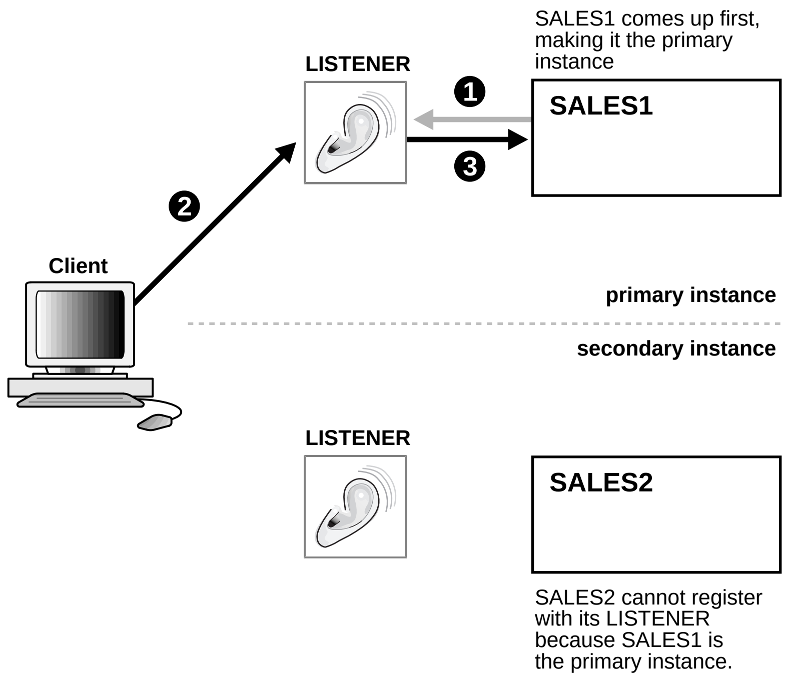
Primary/Secondary Instance in Dedicated Server Environments

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.

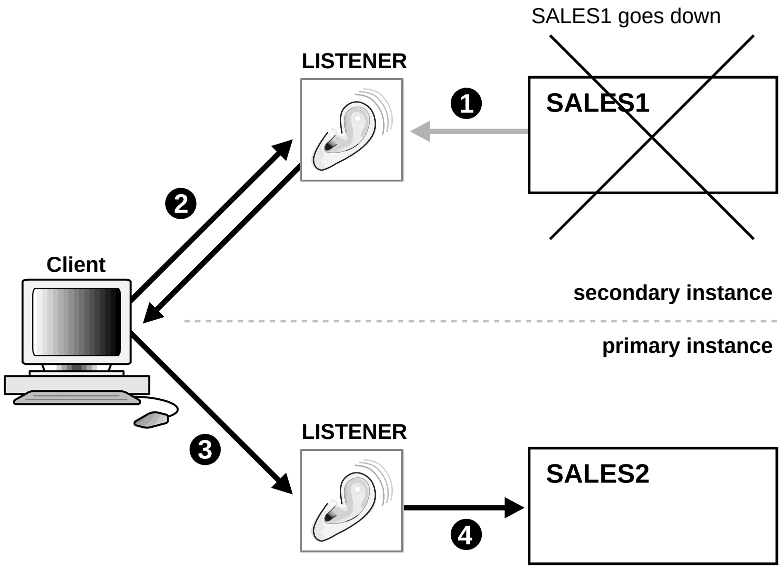
Figure 10–3 Primary/Secondary Instance Feature in Dedicated Server Environments


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.

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



Primary/Secondary Instance Feature and the Shared Server

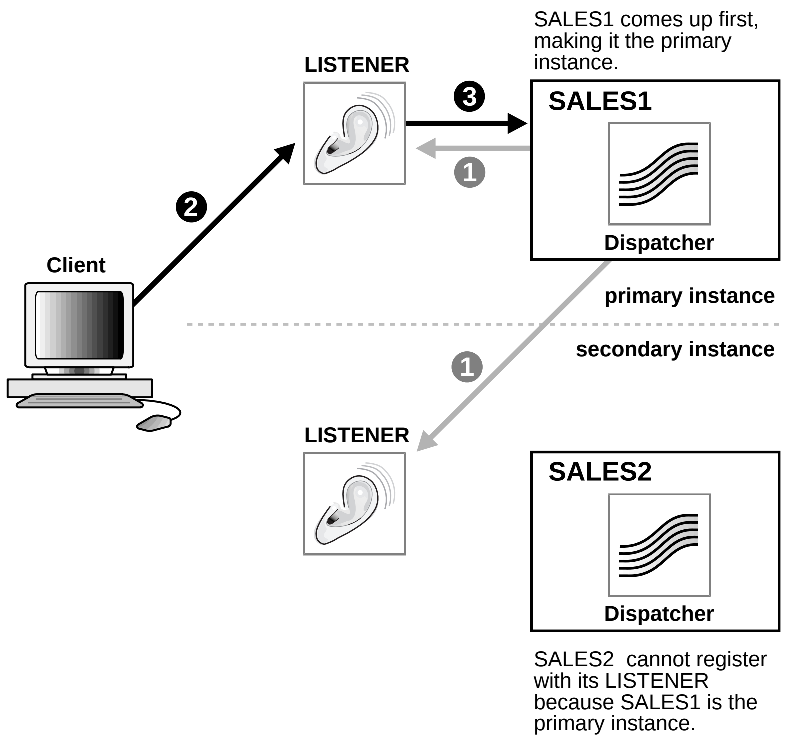
Real Application Clusters provides reconnection performance benefits when running in shared server mode. This is accomplished by the cross-registration of all the dispatchers and listeners in the cluster.

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.

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.

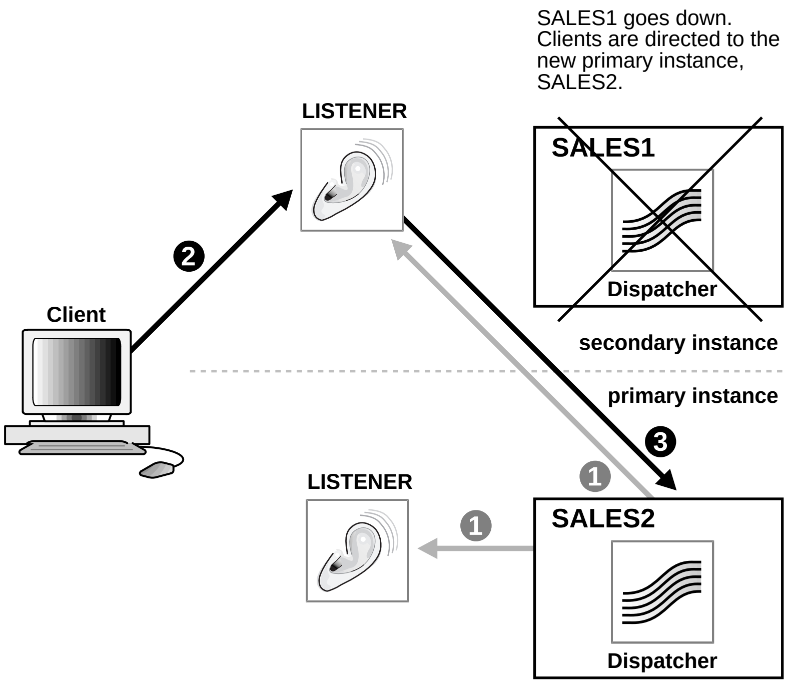
> **See Also:** *Oracle9i Real Application Clusters Installation and Configuration* for instructions on how to connect to secondary instances

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



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.

11

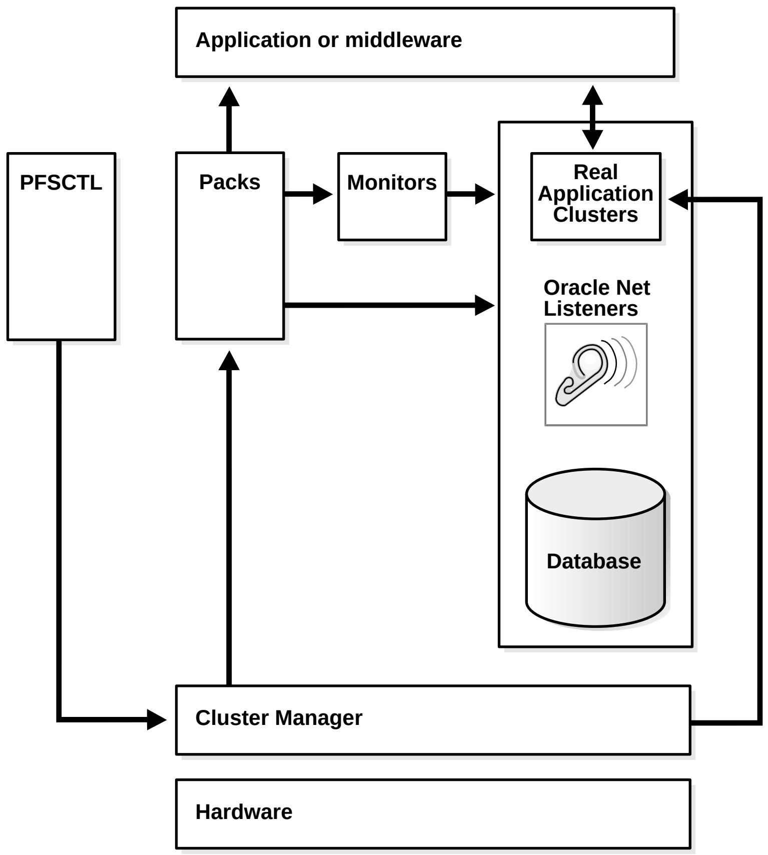
Oracle Real Application Clusters Guard Architecture

This chapter describes the architecture of Oracle Real Application Clusters Guard. It includes the following topics:

Overview of Oracle Real Application Clusters Guard Components

Oracle Real Application Clusters Guard works with Real Application Clusters and the port-specific Cluster Manager to monitor and maintain availability. Figure 11–1 shows the relationship between these components of Oracle Real Application Clusters Guard to the **Cluster Manager (CM)**.

Figure 11–1 How Oracle Real Application Clusters Guard is Related to Real Application Clusters and the Cluster Manager



A database server that runs Real Application Clusters consists of the Oracle database, Real Application Clusters software, and the Oracle Net listeners that accept client requests. These software components run on each node of a cluster.

They use the services provided by the hardware, the operating system, and the port-specific Cluster Manager. The Cluster Manager monitors and reports the health of the nodes in the cluster and controls pack behavior.

A **pack** is a piece or suite of software that adds functionality to **Oracle Enterprise Manager**, ensuring the availability of the set of resources required to run an Oracle **instance**. The pack controls the startup, shutdown, and restarting of Oracle processes. There is one pack for each instance. In the context of Real Application Clusters and Oracle Real Application Clusters Guard, packs were formerly called PFS packs. The application software or middleware receives direction from the packs and from Real Application Clusters.

Oracle Real Application Clusters Guard consists of the components described in the following sections:

- Packs
- PFSCTL Control Utility
- Oracle Real Application Clusters Guard Monitors
- Oracle Real Application Clusters Guard Configuration Templates
- PFS Setup Utility

Packs

The pack controls the startup, shutdown, and restarting of Oracle processes. There is one pack for each instance. Each pack controls the following resources on its node:

- Oracle instance
- Monitors
- Listeners
- IP addresses
- Disk storage, depending on the platform

PFSCTL Control Utility

The `PFSCTL` control script is responsible for starting, stopping, and operating Oracle Real Application Clusters Guard through its interaction with the Cluster Manager. It provides a command-line interface to the user.

See Also: Chapter 12, "Oracle Real Application Clusters Guard Operation"

Oracle Real Application Clusters Guard Monitors

Oracle Real Application Clusters Guard has three monitors:

Instance monitor	Detects termination of the local instance (such as a `SHUTDOWN ABORT`) and initiates a resulting action
Heartbeat monitor	Detects unavailable service (such as an instance hang) for the primary and secondary instance roles and initiates a resulting action
Listener monitor	Monitors and restarts the listeners

See Also: "Monitors" on page 11-9

Oracle Real Application Clusters Guard Configuration Templates

Oracle Real Application Clusters Guard provides configuration templates that allow it to be easily configured. The templates contain configurations for such settings as Oracle Net Services and initialization parameters that have already been tested. The PFS setup utility assists with the generation of the files that are required by Oracle Real Application Clusters Guard. The files are automatically generated with the correct values, derived from the customized templates.

See Also:

- *Oracle Real Application Clusters Guard Administration and Reference Guide*
- Your platform-specific Oracle Real Application Clusters Guard installation guide

PFS Setup Utility

The PFS setup utility assists with the generation of appropriate PFS files for the specified environment, as well as simplified configuration and set up of Oracle Real Application Clusters Guard software. It also makes it easier to deploy changes in the Oracle Real Application Clusters Guard environment.

See Also:

- *Oracle Real Application Clusters Guard Administration and Reference Guide*
- Your platform-specific Oracle Real Application Clusters Guard installation guide

Concepts of Oracle Real Application Clusters Guard

The concepts described in the following sections are important for understanding Oracle Real Application Clusters Guard architecture:

- Instance Roles
- Preferred Primary and Secondary Nodes
- Home and Foreign Nodes

Instance Roles

In a Real Application Clusters environment where the `ACTIVE_INSTANCE_COUNT` parameter in the initialization parameter file (`init.ora`) is set to `1`, an instance has either a **primary instance role** or a **secondary instance role**. The instance that mounts the database first assumes the role of primary instance. The second instance to mount the database assumes the role of secondary instance. If the primary instance fails or is shut down, then the secondary instance automatically assumes the primary instance role. When the failed instance returns to active status, it assumes the role of secondary instance. The `V$INSTANCE` dynamic performance view displays the instance roles of the instances.

See Also: "High Availability Configurations" on page 11-2

Preferred Primary and Secondary Nodes

The **preferred primary node** is the node where the pack with the primary instance role resides by default at startup. It is designated by the user in the Oracle Real Application Clusters Guard configuration file. Oracle Real Application Clusters Guard ensures that the first instance to be started starts on the preferred primary node.

The **preferred secondary node** is the node where the pack with the secondary instance role resides by default at startup. It is designated by the user in the Oracle

Real Application Clusters Guard configuration file. Oracle Real Application Clusters Guard starts the secondary instance on the preferred secondary node.

Real Application Clusters enforces a **Primary/Secondary Configuration** when the `ACTIVE_INSTANCE_COUNT` parameter in the initialization parameter file (`init.ora`) is set to `1`. The user must set `ACTIVE_INSTANCE_COUNT` to `1` as shown in the sample configuration files provided with Oracle Real Application Clusters Guard.

The Oracle Net listener then enforces the routing of work requests to the primary and secondary instances by using the `INSTANCE_ROLE` parameter `tnsnames.ora` found in the `CONNECT_DATA` portion of the `tnsnames.ora` file.

All locks are mastered by the primary instance only. This minimizes communication between nodes and improves performance.

> **See Also:** *Oracle Real Application Clusters Guard Administration and Reference Guide*

Home and Foreign Nodes

The **home node** (primary) is the default node for a specific pack. When the pack is not running on its home node, it is running on its **foreign node** (secondary). At initial startup, each pack runs on its home node.

When a pack runs on its foreign node, the only pack function that occurs is enabling the IP address. New connections that request this IP address are routed to the primary instance by the Oracle Net listener.

Architecture of Oracle Real Application Clusters Guard

Figure 11–2 shows Oracle Real Application Clusters Guard architecture for a two-node cluster. Node A is the primary node, and Node B is the secondary node. Each node contains:

- A Cluster Manager, that executes the run and halt scripts automatically upon failover or as a consequence of a user command
- A pack, that controls the resources available on its node

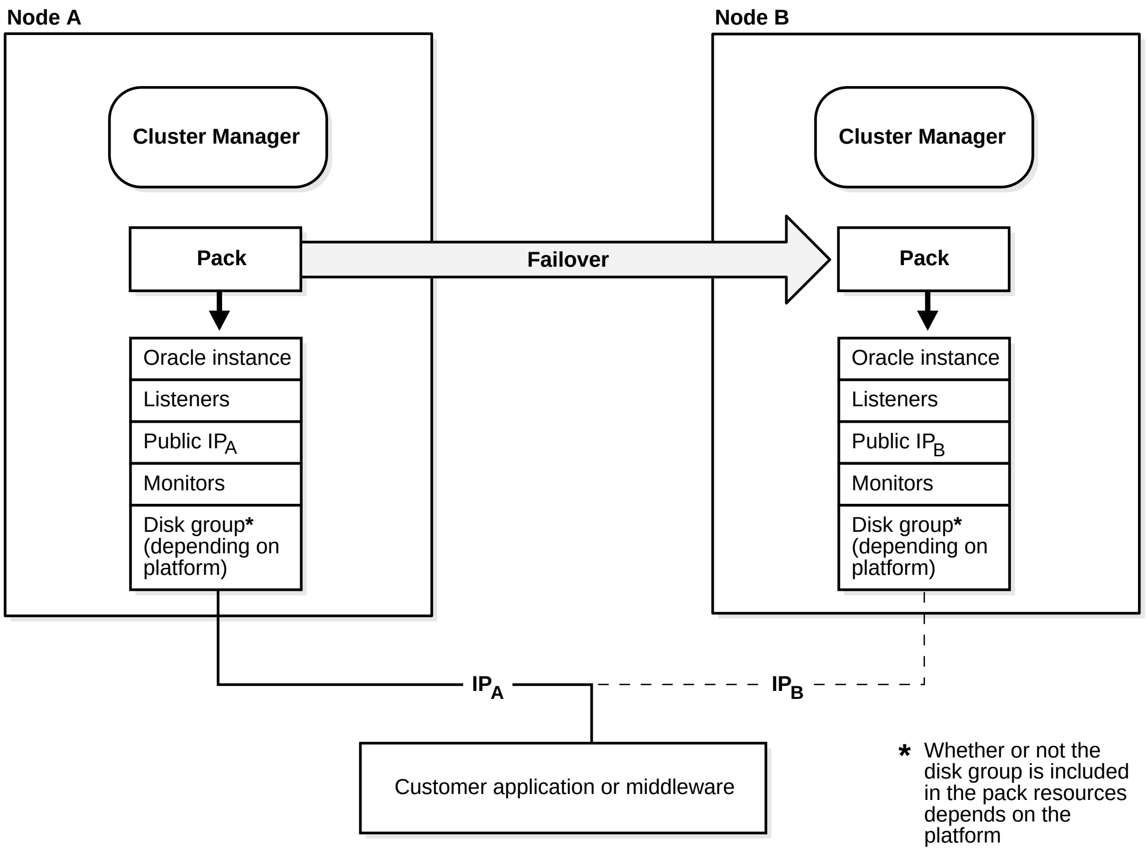
The resources on each node include:

- Oracle instance
- Listeners (both public and private)

- Public IP
- Monitors
- Disk group, depending on the platform

During failover, the primary instance role moves from Node A to Node B, making Node B the new primary node.

Figure 11–2 Oracle Real Application Clusters Guard Architecture for a Two-Node Cluster


Node A
Cluster Manager
Pack
Oracle instance
Listeners
Public IP_A
Monitors
Disk group* (depending on platform)
Failover
Node B
Cluster Manager
Pack
Oracle instance
Listeners
Public IP_B
Monitors
Disk group* (depending on platform)
IP_A
IP_B
Customer application or middleware
* Whether or not the disk group is included in the pack resources depends on the platform

This rest of this section contains the following topics:

- Packs
- Monitors

Packs

A pack is software that ensures the availability of the resources required to run an Oracle instance. It supports and maintains access to the instance through the listeners. A pack controls the startup, shutdown, and restarting of Oracle processes. There is one pack for each instance.

Resources

Each pack controls the following resources on its node:

- Listeners
- IP Addresses
- Disk Storage, depending on the platform
- Monitors

See Also: "Monitors" on page 11-9

Listeners The public listener connects a client to an instance. Private listeners are used by tools such as **Oracle Enterprise Manager** and **Recovery Manager (RMAN)** to connect to an instance. Private listeners can also be used by the database administrator for administration tasks.

IP Addresses A **relocatable IP address** is a public IP address that is started by a Oracle Real Application Clusters Guard pack. A relocatable IP address can be moved among nodes to ensure that it is always available. The IP address is started as the first step when running the pack and is stopped as the last step when halting the pack. This ensures that the address is absent for as little time as possible.

A stationary, private IP address is configured for private tasks such as IPC, heartbeat, system management and RMAN operations. A private listener supports access to the instance through the private IP address.

Disk Storage On platforms that activate and deactivate disk storage without affecting the status of existing disk storage, the pack controls the acquisition and release of disk storage on the home node.

Some platforms affect the status of existing disk storage when they activate and deactivate disk storage. In this case, the Cluster Manager acquires the disk storage in shared mode after the node is brought into the cluster.

Pack Functions

Packs do the following:

- Activate and deactivate disk storage for some platforms
- Start and stop the public IP and public listener
- Start and stop the private listener
- Start and stop the Oracle instance
- Start and stop the Monitors

A pack starts up the Oracle instance and monitors the instance. If it determines that the instance has expired, then it ensures that the resources associated with that instance are moved to the secondary node and reenables service at the secondary node.

A pack can run on either its home node or its foreign node. When it is on its home node, it starts up and shuts down everything. When the pack is on its foreign node, it only starts and stops the IP address. This behavior reduces TCP/IP timeouts for new connections.

Monitors

Oracle Real Application Clusters Guard has three monitors. They are discussed in the following sections:

- Instance Monitor
- Listener Monitor
- Heartbeat Monitor

Instance Monitor

The instance monitor detects termination of the local instance and initiates failover or restarts the instance.

Note: Because the instance monitor connects as a user session, its actions are reflected in database statistics such as enqueue waits.

Listener Monitor

The listener monitor checks and restarts the public and private listeners on its own node. When the public listener fails to restart a configurable number of times within a configurable interval, the listener monitor exits, initiating a halt script. Oracle Real Application Clusters Guard either begins failover or restarts the primary instance, depending on the state of the secondary node.

Heartbeat Monitor

The heartbeat monitor checks the availability of the Oracle instance. The local Oracle instance is considered unavailable if the heartbeat monitor fails to complete after three consecutive attempts. During normal operation, the heartbeat monitor on each instance:

- Updates its own local heartbeat
- Checks the heartbeat of the other instance

The heartbeat monitor on the primary instance also executes a **customer query** specified by the user. Executing the customer query tests whether the primary instance is capable of work.

The heartbeat monitor allows for special circumstances such as instance recovery and unusually large numbers of sessions logging on.

The heartbeat monitor also initiates one kind of failover action: If the primary instance is unavailable and the primary instance role has not resumed normal function on its new node, then the heartbeat monitor initiates takeover. A **takeover** occurs when the secondary node executes failover of the primary instance role to itself.

> **See Also:** *Oracle Real Application Clusters Guard Administration and Reference Guide*

Additional Configurations of Oracle Real Application Clusters Guard

The two-node cluster configuration with one database, with one node serving as the primary node and the other node serving as the secondary node is an ideal configuration. Because it is ideal, users may need to use their resources more efficiently. Multiple installations of Oracle Real Application Clusters Guard can exist on a multinode cluster, so that nodes are shared by several database services. Two multinode configurations have been tested for Oracle Real Application Clusters Guard:

- The **hub configuration**, in which a single node serves as the secondary node to several primary nodes
- The **ring configuration**, in which the nodes serve as primary nodes and also as secondary nodes for other nodes

Although these configurations use resources more efficiently than a configuration with a single dedicated secondary node for each primary node, there are disadvantages to these configurations:

- The ability to isolate failures is reduced.
- Performance may be degraded.

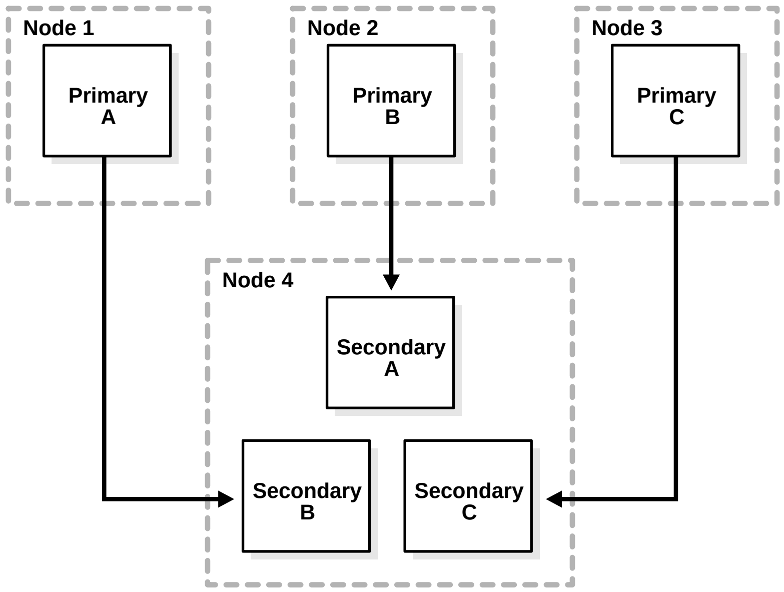
The following sections describe two tested configurations:

- Hub Configuration
- Ring Configuration

Hub Configuration

A hub configuration consists of one node that serves as the secondary node for other nodes that serve as primary nodes for separate installations of Oracle Real Application Clusters Guard databases. The simplest possible hub configuration consists of three nodes. Oracle Real Application Clusters Guard has been tested in a four-node hub configuration. Figure 11–3 shows that the primary instance for database A resides on Node 1, the primary instance for database B resides on Node 2, and the primary instance for database C resides on Node 3. The secondary instances for all three databases reside on Node 4.

Figure 11–3 Four-Node Hub Configuration for Oracle Real Application Clusters Guard Databases



In a stable, (or *resilient*) state, all primary instances run on their preferred primary nodes. When a failure on a primary node occurs, the primary instance fails over to its secondary instance on Node 4. A single failover has minimal impact on the other Oracle Real Application Clusters Guard installations, but if several failures occur, then performance may suffer. In addition, if Node 4 itself fails, then all of the Oracle Real Application Clusters Guard installations lose **resilience**.

Table 11–1 summarizes the advantages and disadvantages of a hub configuration.

Table 11–1 Hub Configuration Advantages and Disadvantages.

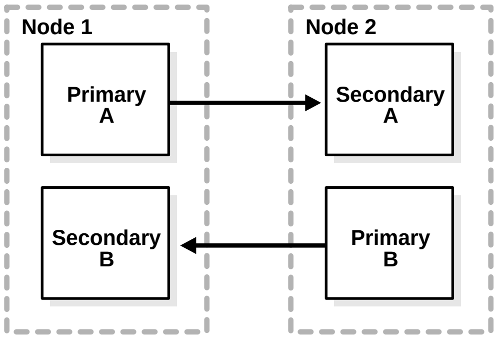
Advantages	Disadvantages
Reduces use of resources: *n*+1 nodes for *n* databases	The secondary node must be capable of handling all of the load for *n* services if all of them fail over at the same time.

Advantages	Disadvantages
	If the secondary node fails, then all of the Oracle Real Application Clusters Guard installations lose their resilience. It is more difficult to isolate failures.

Ring Configuration

Another multinode configuration is the ring configuration. Each node contains a primary instance and serves as the secondary node for another node. The simplest possible ring configuration is the two-node ring configuration shown in Figure 11–4.

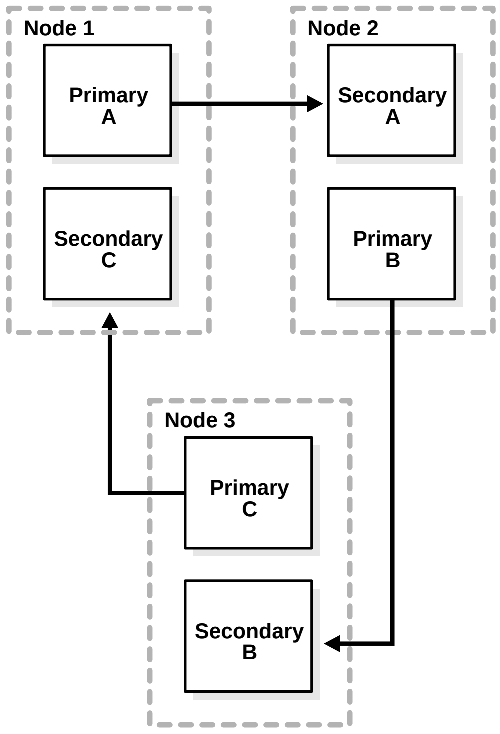
Figure 11–4 Two-Node Ring Configuration for Oracle Real Application Clusters Guard Databases



The primary instance for database A resides on Node 1, while the secondary instance for database A resides on Node 2. The primary instance for database B resides on Node 2, while the secondary instance for database B resides on Node 1.

Oracle Real Application Clusters Guard has been tested for a three-node ring configuration. This is shown in Figure 11–5.

Figure 11–5 Three-Node Ring Configuration for Oracle Real Application Clusters Guard



The primary instance for database A resides on Node 1, while the secondary instance for database A resides on Node 2. The primary instance for database B resides on Node 2, while the secondary instance for database B resides on Node 3. The primary instance for database C resides on Node 3, while the secondary instance for database C resides on Node 1.

Table 11–2 summarizes the advantages and disadvantages of a three-node ring configuration.

Table 11–2 Ring Configuration Advantages and Disadvantages

Advantages	Disadvantages
All nodes hold equal roles.	Failover results in two primary instances sharing a single node. Performance may suffer compared to a dedicated secondary configuration.
More efficient use of resources than hub configuration because there are *n* nodes for *n* databases	It is more difficult to isolate failures compared to a dedicated secondary configuration.
Reduced resource requirement for a single node because only two primary instances must run on a single node.	

12 Oracle Real Application Clusters Guard Operation

This chapter describes the operation of Oracle Real Application Clusters Guard. It contains the following sections:

Overview of Oracle Real Application Clusters Guard Operation

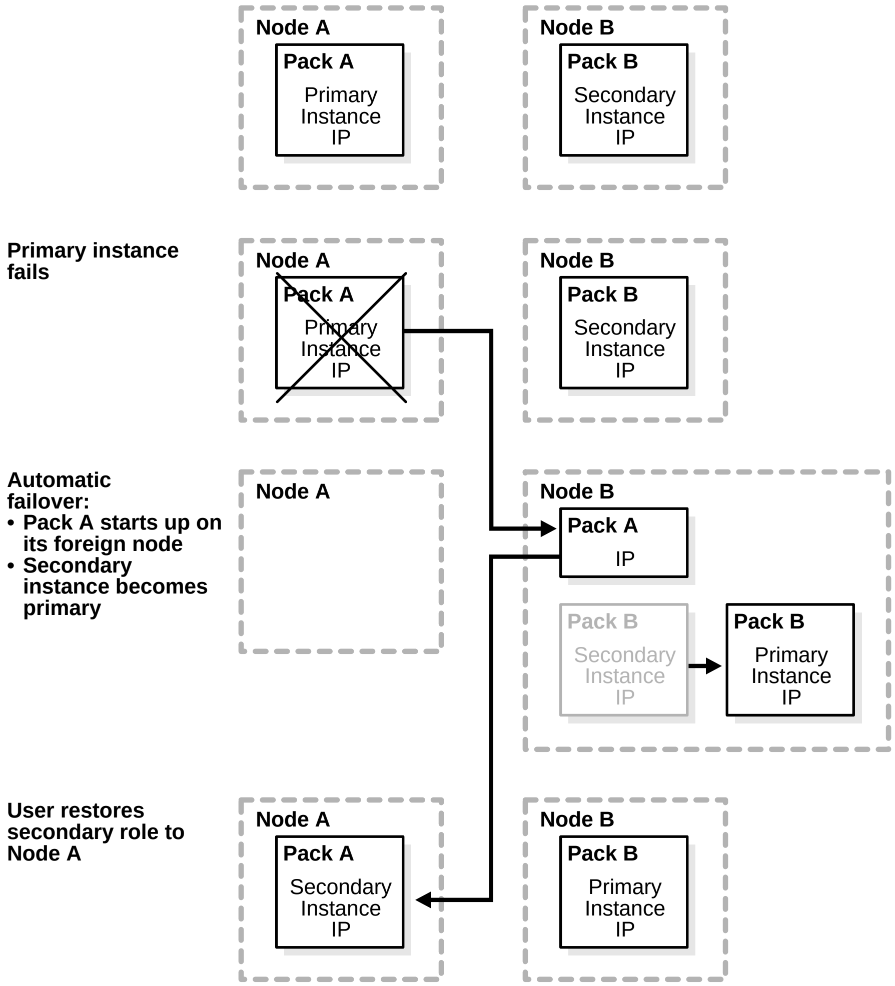
This chapter describes the operation of **Oracle Real Application Clusters Guard**. It is important to distinguish between the automatic actions of Oracle Real Application Clusters Guard and the actions that the user can take when Oracle Real Application Clusters Guard prompts the user. The most typical case is what happens when the **primary instance** fails, and it illustrates the automatic actions of Oracle Real Application Clusters Guard as well as the control that the user has over the final outcome.

> **See Also:** *Oracle Real Application Clusters Guard Administration and Reference Guide* for detailed information about using PFSCTL commands

Failure of the Primary Instance

Figure 12–1 shows what happens when the primary instance fails. During normal operation, both Node A and Node B are operational. Pack A is running on its home node, Node A, and has the **primary instance role**. It contains the primary instance and an IP address. Pack B is running on its home node, Node B, and has the **secondary instance role**. It contains the **secondary instance** and an IP address.

Figure 12–1 Failure of the Primary Instance



If the primary instance fails, then Oracle Real Application Clusters Guard automatically does the following:

- The secondary instance becomes the primary instance.
- Pack A starts on Node B in foreign mode. This means that only its IP address is activated on Node B.

Now both Pack A and Pack B are running on Node B. Pack B contains the primary instance and its IP address. Pack A contains only an IP address. Nothing is running on Node A.

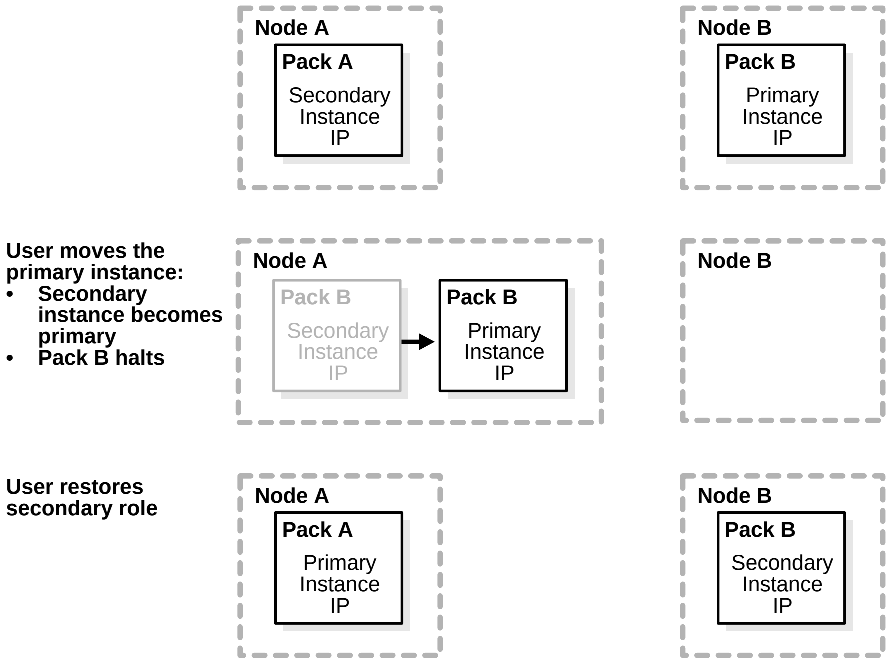
A notification about the failure is sent to the PFS log. If the user has customized the Oracle Real Application Clusters Guard call-home script to notify an administrator of the failure, then the administrator can use the `RESTORE` command to restore the secondary instance role. Oracle Real Application Clusters Guard starts Pack A on Node A. Because the instance on Node B now has the primary instance role, the instance associated with Pack A assumes the secondary instance role when it restarts. When both instances are up and operating, the system has **resilience**.

> **See Also:** *Oracle9i Real Application Clusters Administration* for detailed information about using `PFSCTL` commands and more information about how the call-home feature works in Oracle Real Application Clusters Guard

Restoring the Nodes to their Original Roles

After Oracle Real Application Clusters Guard fails over the primary instance role and the user restores the secondary instance role, the system is resilient and the Packs are on their home nodes, but the instance roles are reversed. If you want the primary instance to run on the preferred primary node, then you must use the `MOVE_PRIMARY` and `RESTORE` commands. Figure 12–2 shows what happens when you return the roles to their preferred nodes.

Figure 12–2 Returning the Packs to Their Home Nodes



Pack A is on Node A and has the secondary instance role. Pack B is on Node B and has the primary instance role. When the user enters the `MOVE_PRIMARY` command, Oracle Real Application Clusters Guard halts Pack B, making the secondary instance, which is running on Node A, becomes the primary instance.

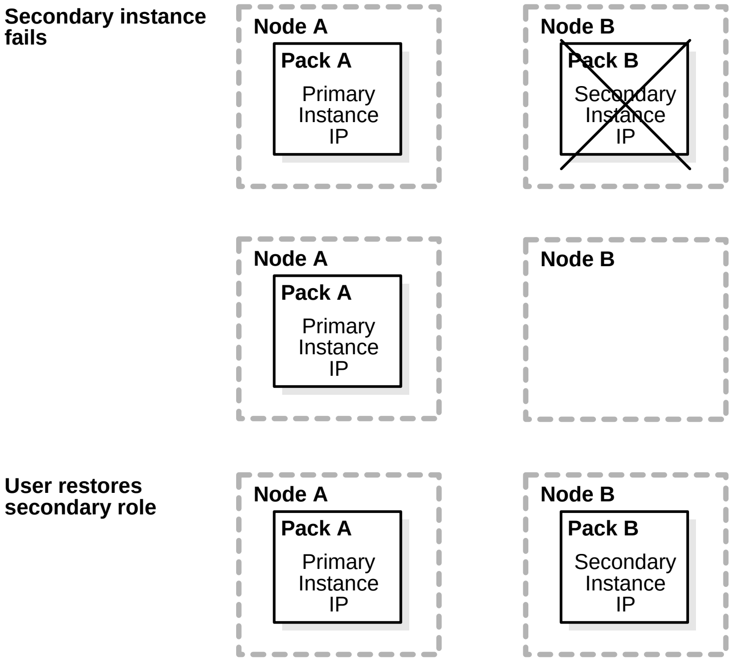
When the user enters the `RESTORE` command, Oracle Real Application Clusters Guard starts Pack B on Node B. Pack B assumes the secondary instance role.

The packs are now running on their home nodes with their original roles.

Failure of the Secondary Instance

Figure 12–3 shows what happens when the secondary instance fails.

Figure 12–3 Failure of the Secondary Instance



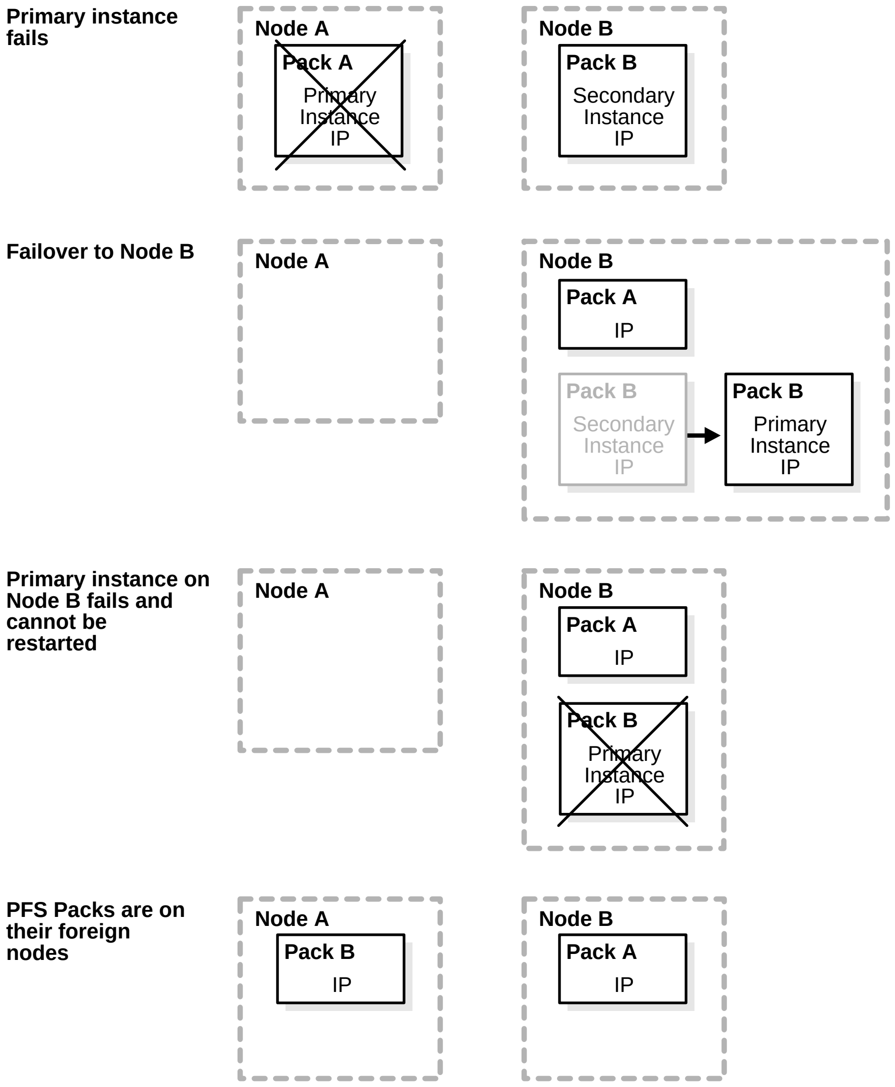
When the secondary instance fails, Oracle Real Application Clusters Guard sends a notification of the failure to the PFS log. The user must enter the `RESTORE` command to restore the secondary instance role to Node B. Both packs retain their original roles on their home nodes.

Note: When there is any failure in the system, it is important to analyze the cause and repair the problem, as well as to restore resilience. For information about troubleshooting, refer to *Oracle9i Real Application Clusters Deployment and Performance.*

Failure of Both Instances

Figure 12–4 shows what happens when both instances fail.

Figure 12–4 Failure of Both Instances



Initially, Pack A is on Node A and has the primary instance role. Pack B is on Node B and has the secondary instance role. If the primary instance fails, then the secondary instance assumes the primary instance role and Pack A starts on Node B with only its IP address active.

If the other instance fails and cannot be restarted, then Pack A remains on Node B and Pack B is started on Node A. Because both packs are on their foreign nodes, only their IP addresses are activated. No instances are running.

Because the Packs are still up, the user must halt them with the `PFSHALT` command. If the user tries to start the instances with the `PFSBOOT` command before halting the packs, then the `PFSBOOT` command will fail.

> **See Also:** *Oracle Real Application Clusters Guard Administration and Reference Guide* for information about how to use `PFSCTL` commands

Part V

Reference

Part Five includes the following reference appendixes:

A
Restrictions

This appendix documents Real Application Clusters compatibility issues and restrictions. Topics in this appendix include:

Compatibility Between Shared and Exclusive Mode

The following sections describe aspects of compatibility between shared and exclusive modes on a Real Application Clusters database:

- Export and Import Utilities
- Mode Compatibility

Export and Import Utilities

The Export utility writes data from an Oracle database into operating system files, and the Import utility reads data from those files back into an Oracle database. This feature of Oracle is the same in shared or exclusive mode.

> ***See Also:*** *Oracle9i Database Utilities* for more information about the Export and Import utilities

Mode Compatibility

Real Application Clusters runs with any Oracle database created in **exclusive (X) mode**. Each instance must have its own set of redo logs.

Oracle in exclusive mode can access a database created or modified by the Real Application Clusters software.

If Real Application Clusters allocates free space to a specific instance, then that space might not be available for inserts for a different instance in exclusive mode. All data in the allocated extents is, of course, always available.

Restrictions

The following sections describe restrictions of Real Application Clusters:

- Maximum Number of Blocks Allocated at a Time
- Restrictions in Cluster Database Mode

Maximum Number of Blocks Allocated at a Time

The `!blocks` option of the `GC_FILES_TO_LOCKS` parameter enables you to control the number of blocks available for use within a free list group. You can use `!blocks` to specify the rate that blocks are allocated within an extent, up to 255 blocks at a time.

Restrictions in Cluster Database Mode

Oracle running multiple instances in cluster database mode supports all the functionality of Oracle in exclusive mode, except as noted in the following sections.

Restricted SQL Statements

In cluster database mode, the following operations are not supported:

- Creating a database (`CREATE DATABASE`)
- Creating a control file (`CREATE CONTROLFILE`)
- Switching the database's archiving mode (the `ARCHIVELOG` and `NOARCHIVELOG` options of `ALTER DATABASE`)

To perform these operations, shut down all instances and start up one instance in exclusive mode.

Maximum Number of Datafiles

The number of datafiles supported by Oracle is operating system-specific. Within this limit, the maximum number allowed depends on the values used in the `CREATE DATABASE` command. This in turn is limited by the physical size of the control file. This limit is the same in cluster database mode as in exclusive mode, but the additional instances of Real Application Clusters restrict the maximum number of files more than a single instance system.

> ***See Also:*** *Oracle9i SQL Reference* and your Oracle operating system-specific documentation

Sequence Number Generators

Real Application Clusters does not support `CACHE ORDER` combination of options for sequence number generators in cluster database mode. Sequences created with both of these options are ordered but not cached when running in Real Application Clusters.

Free Lists with Import and Export Utilities

The Export utility does not preserve information about multiple free lists and free list groups. The metadata of the table where it is imported contains the free list and free list group information that is henceforth associated with the data blocks.

Therefore, if you use the Export and Import utilities to back up and restore your data, then it will be difficult to import the data so that it is partitioned again.

B

Using Pre-Release 1 (9.0.1) Multi-Block Lock Assignments (Optional)

This appendix describes configuring locks that were used before Release 1 (9.0.1) to cover multiple blocks. This methodology has been largely superseded by the automated resource management capabilities provided by **Cache Fusion** in Release 1 (9.0.1). However, overriding Cache Fusion to use pre-Release 1 (9.0.1) locks could be desirable in rare cases. Topics in this appendix include:

- When to Use Pre-Release 1 (9.0.1) Lock Setting
- How to Use Pre-Release 1 (9.0.1) Lock Setting
- Lock Granularity
- Understanding Lock Management

When to Use Pre-Release 1 (9.0.1) Lock Setting

The methodology described in this appendix has been largely superseded by **Cache Fusion** in Release 1 (9.0.1). Overriding Cache Fusion to do lock setting is only desirable in Real Application Clusters installations that have a large number of read-mostly blocks accessed by an application in a partitioned manner. In nearly all other cases, it is recommended that users do *not* override the automated **Cache Fusion** resources. This is because using pre-Release 1 (9.0.1) locks, you lose the inherent advantages of Cache Fusion transfers. Without Cache Fusion, the **Global Cache Service (GCS)** is implemented with costly **forced disk write**s.

How to Use Pre-Release 1 (9.0.1) Lock Setting

You can override Cache Fusion and assign locks-to-files by setting initialization parameters to allocate the number of locks desired. Automated Cache Fusion concurrency control is active by default unless you set the `GC_FILES_TO_LOCKS` parameter.

Processing Locks to Ensure Optimal Performance

Locks can manage one or more blocks of any class: data blocks, undo blocks, segment headers, and so on. However, a given **Global Cache Service (GCS)** resource can manage only one block class.

With pre-Release 1 (9.0.1) lock setting, cache coherency is ensured by forcing requesting instances to acquire locks from instances that hold locks on data blocks before modifying or reading the blocks. Pre-Release 1 (9.0.1) lock setting also allows only one instance at a time to modify a block. If a block is modified by an instance, then the block must first be written to disk before another instance can acquire the lock and modify it.

Locks use a minimum amount of communication to ensure cache coherency. The amount of cross-instance activity—and the corresponding performance of Real Application Clusters—is evaluated in terms of *forced writes*. A forced write is when one instance requests a block that is held by another instance. To resolve this type of request, Oracle writes (or **ping**s) the block to disk so the requesting instance can read it in its most current state. In Oracle9*i*, Cache Fusion allows reading of dirty **past image (PI)** blocks across instances.

Heavily loaded applications can experience significant enqueuing activity, but they do not necessarily have excessive forced writes. If data is well partitioned, then the enqueuing is local to each node, and therefore forced writing does not occur.

See Also: *Oracle9i Real Application Clusters Installation and Configuration* for detailed information about allocating locks

Lock Granularity

There are two levels of lock granularity discussed in this section:

- 1:1 Locks
- 1:n Locks

1:1 Locks

A 1:1 lock means one lock for each block.

1:*n* Locks

1:*n* locks implies that a lock manages two or more data blocks as defined by the value for *n*. With 1:*n* locks, a few locks can manage many blocks and thus reduce lock operations. For read-only data, 1:*n* locks can perform faster than 1:1 locks during certain operations such as parallel execution. The `GC_FILES_TO_LOCKS` parameter is only useful to get 1:*n* lock granularity

If you partition data according to the nodes that are most likely to modify it, then you can implement disjoint lock sets, each set belonging to a specific node. This can significantly reduce lock operations. 1:*n* locks are also beneficial if you have a large amount of data that a relatively small number of instances modify. If a lock is already held by the instance that modifies the data, no lock activity is required for the operation.

See Also: *Oracle9i Real Application Clusters Deployment and Performance* for detailed information about configuring Global Enqueue Service enqueues

Understanding Lock Management

The number of locks assigned to datafiles and the number of data blocks in those datafiles determines the number of data blocks managed by a single lock.

When the GC_FILES_TO_LOCKS parameter *is* set for a file, then the number of blocks for each lock can be expressed as follows for each file level. This example assumes values of GC_FILES_TO_LOCKS = 1:300,2:200,3-5:100.

$$\text{File 1: } \frac{\textit{file1 blocks}}{\text{300 locks}}$$

$$\text{File 2: } \frac{\textit{file2 blocks}}{\text{200 locks}}$$

$$\text{File 3: } \frac{\text{sum } \textit{(file3, file4, file5 blocks)}}{\text{100 locks}}$$

If the size of each file, in blocks, is a multiple of the number of locks assigned to it, then each 1:*n* lock manages exactly the number of data blocks given by the equation.

If the file size is *not* a multiple of the number of locks, then the number of data blocks for each 1:*n* lock can vary by one for that datafile. For example, if you assign 400 locks to a datafile that contains 2,500 data blocks, then 100 locks manage 7 data blocks each and 300 locks manage 6 blocks. Any datafiles not specified in the GC_FILES_TO_LOCKS initialization parameter use the remaining locks.

If *n* files share the same 1:*n* locks, then the number of blocks for each lock can vary by as much as *n*. If you assign locks to individual files, either with separate clauses of GC_FILES_TO_LOCKS or by using the keyword EACH, then the number of blocks for each lock does not vary by more than one.

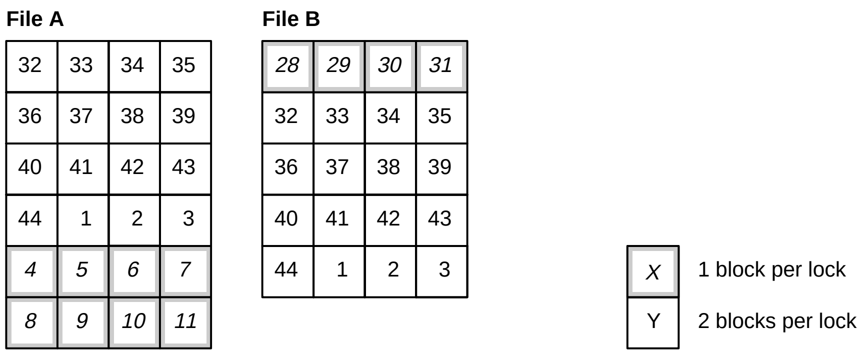
If you assign 1:*n* locks to a set of datafiles collectively, then each lock usually manages one or more blocks in each file. Exceptions can occur when you specify contiguous blocks (by using the `!blocks` option) or when a file contains fewer blocks than the number of locks assigned to the set of files.

> **See Also:** *Oracle9i Real Application Clusters Deployment and Performance* for details on how to use the `GC_FILES_TO_LOCKS` parameter

Example of Locks Covering Multiple Blocks

The following illustrates how 1:*n* locks can manage multiple blocks in different files. Figure B–1 assumes 44 locks assigned to two files that have a total of 44 blocks. `GC_FILES_TO_LOCKS` is set to A,B:44

Figure B–1 Locks Covering Blocks in Multiple Files



Block 1 of a file does not necessarily begin with lock 1. A hashing function determines which lock a file begins with. In file A that has 24 blocks, block 1 hashes to lock 32. In file B that has 20 blocks, block 1 hashes to lock 28.

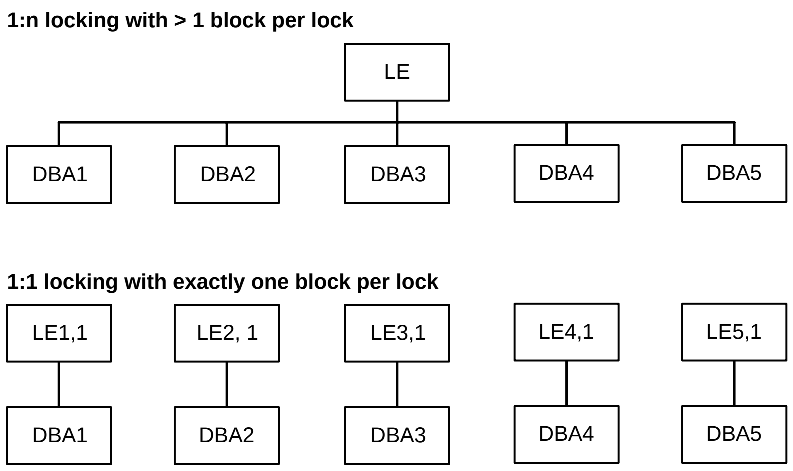
In Figure B–1, locks 32 through 44 and 1 through 3 are used to manage two blocks each. Locks 4 through 11 and 28 through 31 manage one block each; and locks 12 through 27 manage no blocks at all.

In a worst case scenario, if two files hash to the same lock as a starting point, then all the common locks will manage two blocks each. If your files are large and have multiple blocks for each lock (about 100 blocks for each lock), then this is not an important issue.

Lock Element to Data Block Correspondence

Figure B–2 illustrates the correspondence of a **global cache element** to blocks. A global cache element is an Oracle-specific data structure representing lock set with the pre-Release 1 (9.0.1) methodology. There is a one-to-one correspondence between a global cache element and a lock.

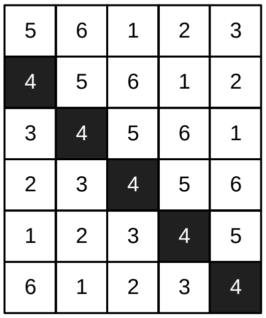
Figure B–2 *1:n Locking and 1:1 Locking*


1:n locking with > 1 block per lock
LE
DBA1
DBA2
DBA3
DBA4
DBA5
1:1 locking with exactly one block per lock
LE1,1
LE2, 1
LE3,1
LE4,1
LE5,1
DBA1
DBA2
DBA3
DBA4
DBA5

Periodicity of Locks

You should also consider the *periodicity* of locks. Figure B–3 shows a file of 30 blocks that is managed by six locks. The example file shown has 1:*n* locks set to begin with lock number 5. As suggested by the shaded blocks managed by lock number 4, use of each lock forms a pattern over the blocks of the file.

Figure B–3 Periodicity of Locks



Forced Writing: Signaling the Need to Update

In Real Application Clusters, a particular data block can only be modified by one instance at a time. If one instance modifies a data block that another instance needs, whether forced writing is required depends on the type of request submitted for the block.

If the requesting instance needs the block for modification, then the holding instance's locks on the data block must be converted accordingly. The first instance must accomplish a **forced disk write** before the requesting instance can read it.

The **LCK process** signals a need between the two instances. If the requesting instance only needs the block in **consistent read (CR)** mode, the lock process of the holding instance transmits a CR version of the block to the requesting instance by way of the interconnect. In this scenario, forced writing is much faster.

Data blocks are only force-written when a block held in exclusive current (XCUR) state in the buffer cache of one instance is needed by a different instance for modification. In some cases, therefore, the number of locks covering data blocks might have little effect on whether a block gets force-written.

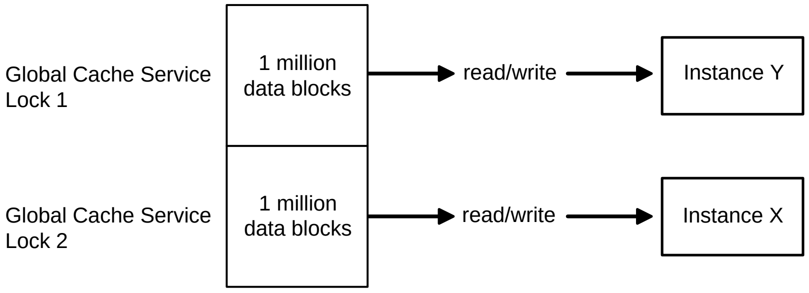
An instance can relinquish an exclusive lock on a block and still have a row lock on rows in it: forced writing is independent of whether a commit has occurred. You can modify a block, but whether it is force-written is independent of whether you have made the commit.

Partitioning to Avoid Forced Writing

If you have partitioned data across instances and are doing updates, then your application can have, for example, one million blocks on each instance. Each block is managed by one lock yet there are no forced reads or forced writes.

As shown in Figure B–4, assume a single lock manages one million data blocks in a table and the blocks in that table are read from or written into the **System Global Area (SGA)** of instance X. Assume another single lock manages another million data blocks in the table that are read or written to the SGA of instance Y. Regardless of the number of updates, there will be no forced reads or forced writes on data blocks between instance X and instance Y.

Figure B–4 Partitioning Data to Avoid Forced Writing



See Also: *Oracle9i Real Application Clusters Deployment and Performance* for more information about partitioning applications to avoid forced writing

Glossary

active

An **instance** that shows activity. Instance activity is monitored by means of a **heartbeat**.

agent

In the client/server model, the part of the system that performs information preparation and exchange on behalf of a client or server. Especially in the phrase *intelligent agent* it implies an automatic **process** that can communicate with other agents to perform some collective tasks on behalf of one or more users.

acquisition interrupt

An acquisition interrupt is a software synchronization feature used for notification. Also known as a **wake up call**. Real Application Clusters also uses a **blocking interrupt**.

ARCH

See **archive (ARCH)**

archive (ARCH)

The archive background **process** creates a single file containing one or (usually) more separate files plus information to allow them to be extracted (separated) by Real Application Clusters.

blocking interrupt

A blocking interrupt is a software synchronization feature that notifies the process holding the access right to a resource that another process needs to access the same resource in an incompatible mode. (The **shared (S) mode** and **exclusive (X) mode**, for example, are incompatible). See also **acquisition interrupt**.

buffer state

The buffer state is the state of a buffer in the local cache of an instance.

cache coherency

The synchronization of data in multiple caches so that reading a memory location through any cache will return the most recent data written to that location through any other cache. Sometimes also called *cache consistency*.

Cache Fusion

A *diskless* cache coherency mechanism, used in Real Application Clusters, that provides copies of blocks directly from the holding instance's memory cache to the requesting instance's memory cache. Cache Fusion is the collective term for the group of features that were phased in starting with Oracle8*i*.

call

A call is to operate a command in order to execute software functionality.

central processing unit

See **CPU**

CFVRR

See **control file voting results record (CFVRR)**

CGS

See **cluster group services (CGS)**

checkpoint

A checkpoint is the conclusion of the process of writing a buffer of a **dirty block** to stable **storage**.

Client Load Balancing Cluster

See **cluster**

cluster

A cluster is a highly available combination of hardware and software. In a cluster, instances typically run on different nodes that coordinate with one another when accessing the shared database residing on disk.

cluster database

A cluster database is the generic term for a database configured and controlled by the **Real Application Clusters** product.

cluster group services (CGS)

The software layer that provides mechanisms for instances to exchange configuration information, interprocess communication (IPC) port identifiers, and other kinds of meta data needed for Real Application Clusters operation. Cluster group services are also known as LMON-provided services.

Cluster Manager (CM)

Cluster Manager is an Operating System-Dependent component that discovers and tracks the membership state of **node**s by providing a common view of cluster membership across the **cluster**. CM monitors **process** health. The LMON process, a background process that monitors the health of the **Global Cache Service (GCS)**, registers and de-registers from the CM. The CM also manages recovery from any network card or cable failures.

CM

See **Cluster Manager (CM)**

connect-time failover

See **failover**

consistent read (CR)

The **Global Cache Service (GCS)** ensures that a consistent read block (also known as the master copy data block) is maintained. The consistent read block is the master block version that holds all the changes. It is held in at least one **System Global Area (SGA)** in the cluster if the block is to be changed. If an instance needs to read it, then the current version of the block may reside in many buffer caches as a shared resource. Thus, the most recent copy of the block in all System Global Areas contains all changes made to that block by all instances, regardless of whether any transactions on those instances have committed.

contention

Competition for resources. The term is used in networks to describe a situation where two or more nodes attempt to access the same resource at the same time. Contention is resolved by concurrency control mechanisms.

control file voting results record (CFVRR)

Part of the recovery process. The CFVRR is a file record created by the **instance membership recovery (IMR)** arbiter after it settles the membership *votes* of nodes in a **cluster**.

cooked partition

A cooked partition is a portion of a physical disk where an extended partition is created, logical partitions are assigned, and formatting has been completed. In contrast, a unformatted partitions is called a **raw partition**.

CPU

The part of a computer that controls all the other parts. A parallel computer has several CPUs that can share other resources such as memory and peripheral devices.

CR

See **consistent read (CR)**

customer query

A **PL/SQL** procedure containing a query that should represent the actual work that must be done in the **instance**. The purpose of the customer query is to determine whether the primary instance is capable of work. The customer modifies the PL/SQL procedure that is provided in the **Oracle Real Application Clusters Guard** `catpfs.sql` script.

daemon

Abbreviation for disk and execution monitor. A program that is not invoked explicitly, but lies dormant waiting for specific conditions to occur.

database administrator (DBA)

An individual responsible for the design and management of the database and for the evaluation, selection and implementation of the database management system. In smaller organizations, the data administrator and database administrator are often the same individual. However, when they are different, the database administrator's function is more technical. The database administrator would implement the database software that meets the requirements outlined by the organization's data administrator and systems analysts. DBA is also an acronym for data block address.

Database Configuration Assistant (DBCA)

A tool for creating and deleting databases.

Database Mount Lock

The Database Mount Lock shows whether an **instance** has mounted a particular database in exclusive or shared mode. This lock is only used with Real Application Clusters. The Database Mount Lock is the only multi-instance lock used with Real Application Clusters in exclusive mode. It prevents another instance from mounting the database in shared or exclusive modes.

database writer (DBWR)

The database writer (DBWR) is a background process **daemon** that writes new data (modified blocks) from the **System Global Area (SGA)** to the datafiles. Each **instance** has a DBWR. The DBWR is optimized to minimize disk writes. Generally disk writes occur only when more data needs to be read into the SGA and there is not enough free space in the database buffers. The least recently used data is written to the datafiles first.

Data Definition Language (DDL)

A language enabling the structure and instances of a database to be defined in a human-readable, and machine-readable form.

The **structured query language (SQL)** contains DDL commands that can be used either interactively or within programming language source code to define databases and their components, such as the `CREATE` and `DROP` statements.

Data Dictionary Cache

See **dictionary cache**

datafile

A datafile is a physical operating system file on disk that was created by Oracle. It contains data structures such as tables and indexes. A datafile can only belong to one database.

Data Guard

See **Oracle9i Data Guard**

DBA

See **database administrator (DBA)**

DBCA

See **Database Configuration Assistant (DBCA)**

DBWR

See **database writer (DBWR)**

DDL

See **Data Definition Language (DDL)**

Decision Support System (DSS)

Software tools that help with decision support.

diagnosability daemon

The diagnosability **daemon** is a Real Application Clusters background **process** that captures diagnostic data on **instance** process failures. No user control is required for this daemon.

dictionary cache

The data dictionary cache contains information from the data dictionary, the metadata store.

dirty block

A data block that has been changed.

disk and execution monitor

See **daemon**

DSS

See **Decision Support System (DSS)**

dynamic resource remastering

Dynamic resource remastering is the ability to move the ownership of a resource between instances of Real Application Clusters. Dynamic resource remastering is used to implement **resource affinity** for increased performance. Resource remastering is deferred when instances enter or leave the **cluster**.

EMCA

See **Enterprise Manager Configuration Assistant (EMCA)**

enqueue

Enqueues are shared memory structures that serialize access to database resources. Enqueues are local to one **instance** if Real Application Clusters is not enabled. When you enable Real Application Clusters, enqueues can be global to a database. (See also: **latch**, **lock**, and **resource**.)

Enterprise Manager

See **Oracle Enterprise Manager**

Enterprise Manager Configuration Assistant (EMCA)

A tool for creating, deleting, and modifying Enterprise Manager configurations and settings.

exclusive current (XCUR)

The buffer state name for an exclusive resource.

exclusive (X) mode

A write-only global resource mode. In this mode no other access is allowed. See also: **null (N) mode** and **shared (S) mode**.

extent

An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

extent allocation

The **process** used to assign a number of contiguous data blocks.

failover

The means of failure recognition and recovery used by Real Application Clusters.

fault tolerance

The ability of a system or component to continue normal operation despite the presence of hardware or software faults. This normally involves some degree of redundancy.

FDDI

See **Fiber Distributed Data Interface (FDDI)**

Fiber Distributed Data Interface (FDDI)

A standard of the American National Standards Institute (ANSI) for a 100 Mb per second local area network architecture. The underlying medium is often optical fiber and the topology is a dual-attached, counter-rotating token ring.

Fibre Channel

Fibre Channel is the generic term for a high speed serial data transfer architecture recently standardized by the American National Standards Institute (ANSI). The Fibre Channel architecture was developed by the Fibre Channel Industry Association (FCIA), a consortium of computer and mass storage manufacturers. The best-known Fibre Channel standard is the Fibre Channel Arbitrated Loop (FC-AL). (See also: **Storage Area Network (SAN)**.

five nines availability

A colloquial term for 99.999% system availability.

forced disk write

In Real Application Clusters, a particular data block can only be modified by one **instance** at a time. If one instance modifies a data block that another instance needs, then whether a forced disk write is required depends on the type of request submitted for the block. If the requesting instance needs the block for modification, then the holding instance's resources on the data block must be converted accordingly. The first instance must write the block to disk before the requesting instance can read it. This constitutes the forced disk write to a block. A forced disk write is also often called a **ping**. Most forced disk writes have been eliminated by **Cache Fusion**, because it provides copies of blocks directly from the holding instance's memory cache to the requesting instance's memory cache.

foreign node

The node where a runs when it its not running on its default (**home node**). When a pack is running on its foreign (secondary) node, only the IP address is enabled.

free list group

A free list group is a set of free lists available for use by one or more instances.

global cache element

A global cache element is an Oracle-specific data structure representing a **Cache Fusion** resource. There is a 1:1 corresponding relationship between a global cache element and a Cache Fusion resource in the **Global Cache Service (GCS)**.

Global Cache Service (GCS)

The Global Cache Service is the controlling process that implements **Cache Fusion**. It maintains the block mode for blocks in the global role. It is responsible for block transfers between instances. The Global Cache Service employs various background processes such as the **Global Cache Service Processes (LMSn)** and **Global Enqueue Service Daemon (LMD)**.

Global Cache Service Processes (LMS*n*)

The Global Cache Service Processes (LMS*n*) are the processes that handle remote **Global Cache Service (GCS)** messages. Current Real Application Clusters software provides for up to 10 Global Cache Service Processes. The number of LMS*n* varies depending on the amount of messaging traffic among nodes in the **cluster**. The LMS*n* handle the **acquisition interrupt** and **blocking interrupt** requests from the remote instances for Global Cache Service resources. For cross-instance consistent read requests, the LMS*n* will create a consistent read version of the block and send it to the requesting instance. The LMS*n* also control the flow of messages to remote instances.

Global Enqueue Service (GES)

This service coordinates enqueues that are shared globally.

Global Enqueue Service Daemon (LMD)

The Global Enqueue Service Daemon (LMD) is the resource **agent** process that manages **Global Enqueue Service (GES)** resource requests. The LMD **process** also handles deadlock detection **Global Enqueue Service (GES)** requests. Remote resource requests are requests originating from another **instance**. (See also: **daemon**.)

Global Enqueue Service Monitor (LMON)

The background Global Enqueue Service Monitor (LMON) monitors the entire **cluster** to manage global resources. LMON manages **instance** and **process** expirations and the associated recovery for the **Global Cache Service (GCS)** and **Global Enqueue Service (GES)**. In particular, LMON handles the part of recovery associated with global resources. LMON-provided services are also known as **cluster group services (CGS)**.

Global Resource Directory

The data structures associated with global resources. It is distributed across all instances in a **cluster**.

Global Services Daemon (GSD)

The Global Services Daemon (GSD) is a component that receives requests from **SRVCTL** to execute administrative job tasks, such as startup or shutdown. The command is executed locally on each node, and the results are sent back to SRVCTL. The **daemon** is installed on the nodes by default. It should not be deleted.

Group Membership Service (GMS)

See **cluster group services (CGS)**

GSD

See **Global Services Daemon (GSD)**

HA

See **high availability**

hardware failover

Hardware failover refers to **failover** performed by the platform-specific **Cluster Manager (CM)**. If a node or the **instance** running on it fails, the cluster manager restarts the instance on another node in the **cluster**. Restarting the Oracle instance requires moving the IP addresses, volumes, and file systems containing the Oracle datafiles. It also requires starting the Oracle server and opening the datafiles on the new node.

heartbeat

A periodic message that shows that an **instance** is **active**.

high availability

High availability refers to systems with redundant components that provide consistent and uninterrupted service, even in the event of hardware or software failures. This involves some degree of redundancy. Availability is often expressed as a percentage of time that the database is available over the period of a year, such as 99.95%. It can also be expressed as the number of hours times the number of days in the week that the system is expected to run, such as 24 hours a day and 7 days a week. High availability can be defined to exclude unplanned downtime only or both planned and unplanned downtime.

high water mark

The high water mark is the boundary between used and unused space in a segment.

home node

The default (primary) node for a specific **pack**. At initial startup, each pack runs on its home node. (See also: **foreign node**.)

hub configuration

A configuration in which a single node serves as the secondary node to several primary nodes.

IMR

See **instance membership recovery (IMR)**

INDX

`INDX` is the default database tablespace that stores indexes associated with the data in the `USER` tablespace.

initsid.ora

The `initsid.ora` file is an initialization file that contains parameters unique for an **instance** and points to `init`*`dbname`*`.ora` for database parameters.

initdbname.ora

The `init`*`dbname`*`.ora` file is an initialization file that contains database parameters.

Input/Output

See **I/O**.

instance

For a Real Application Clusters database, each node within the **cluster** has an instance of the running Oracle software referencing the database.

When a database is started on a database server (regardless of the type of system), Oracle allocates a memory area called the **System Global Area (SGA)** and starts one or more Oracle processes. This combination of the SGA and the Oracle **process** is called an instance. The memory and processes of an instance manage the associated database's data efficiently and serve the one or more users of the database. You can connect to any instance to access any information that resides within a Real Application Clusters database.

Each instance has unique **system identifier (SID)**, instance name, rollback segments, and **thread** ID.

instance membership recovery (IMR)

Instance membership recovery (IMR) is the method used by Real Application Clusters guaranteeing that all **cluster** members are functional or **active**. IMR polls and arbitrates the membership. Any members that do not show a heartbeat in the control file record or who do not respond to periodic status queries are presumed to have expired. The Instance Membership Recovery arbiter settles the membership *votes* of nodes in a cluster, and creates a **control file voting results record (CFVRR)**.

instance name

The instance name represents the name of the **instance** and is used to uniquely identify a specific instance when multiple instances share common services names. The instance name is identified by the `INSTANCE_NAME` parameter in the **initsid.ora** file.

The instance name is the same as the **system identifier (SID)**.

You can use various **structured query language (SQL)** options with the `INSTANCE_NUMBER` initialization parameter to associate extents of data blocks with instances.

The instance number is depicted by the `INSTANCE_NUMBER` parameter in the instance initialization file, `initsid.ora`.

interconnect

The communication link between the nodes.

interinstance contention

See **contention**

interprocess communication (IPC)

Interprocess communication (IPC) is a high speed Operating System-Dependent transport component. IPC reliably transfers messages between instances on different nodes.

interrupt

See **acquisition interrupt** and **blocking interrupt**

I/O

In the context of Real Application Clusters, I/O is an Operating System-Dependent component that provides I/O to access shared disks.

IPC

See **interprocess communication (IPC)**

kernel

In the context of databases: The set of foreground and background processes that implement a database.

In the context of languages: An essential subset of a programming language, where other constructs are (or could be) defined. Also known as a core language.

In the context of operating systems: The essential part of Unix or other operating systems, responsible for resource allocation, low-level hardware interfaces, security, and so on.

latch

A latch is a simple, low-level serialization mechanism that protect in-memory data structures in the **System Global Area (SGA)**. Latches do not protect datafiles, are automatic, and are held for a very short time in exclusive mode. Because latches are synchronized within a node, they do not facilitate internode synchronization. (See also: **enqueue**, **lock**, and **resource**.)

LCK process

The LCK **process** manages **instance** global enqueue requests and cross-instance call operations. Workload is automatically shared and balanced when there are multiple **Global Cache Service Processes (LMSn)**.

library cache invalidation

Objects in the library cache can become invalid due to object dependencies. The object is recompiled on its next use.

listener

The listener is a separate **process** that resides on the server whose responsibility is to listen for incoming client connection requests and manage the traffic to the server.

The listener brokers the client request, handing off the request to the server. Every time a client (or server acting as a client) requests a network session with a server, a listener receives the actual request. If the client's information matches the listener's information, then the listener grants a connection to the server.

listener.ora

A configuration file for the **listener** that identifies the listener name, protocol addresses that it is accepting connection requests on, and Services it is listening for.

The `listener.ora` file typically resides in the `$ORACLE_HOME/network/admin` directory on UNIX platforms and the `ORACLE_HOME\network\admin` directory on Windows NT.

Oracle databases do not require identification of the database service in listener.ora because of **service registration**. However, static service configuration is required for an Oracle databases if you plan to use a listener.

LMD

See **Global Enqueue Service Daemon (LMD)**

LMON

See **Global Enqueue Service Monitor (LMON)**

LMS*n*

See **Global Cache Service Processes (LMSn)**

local enqueues

Synchronization mechanisms utilized to coordinate concurrent access to shared data structures in **cluster** databases. Local enqueues are called **local locks** in single **instance** Oracle. (See also: **enqueue**.)

local locks

In single **instance** Oracle, local locks are synchronization mechanisms utilized to coordinate concurrent access to shared data structures. In a **cluster database**, local locks are called local enqueues.

lock

See **enqueue**, **latch**, and **resource**

LCK process

The LCK **process** manages **instance** resource requests and cross-instance call operations. Workload is automatically shared and balanced when there are multiple **Global Cache Service Processes (LMSn)**.

Massively Parallel Processing (MPP)

Computing that uses many separate CPUs running in parallel to execute a single program.

master free list

A list of blocks containing available space drawn from any extent in a table.

mastering

See **resource mastering**

mean time between failures (MTBF)

The average time (usually expressed in hours) that a component works without failure. It is calculated by dividing the total number of failures into the total number of operating hours observed. The term can also mean the length of time a user can reasonably expect a device or system to work before an failure occurs.

mean time to failure (MTTF)

The average period of time that a component will work until failure.

mean time to recover (MTTR)

The average time that it takes to get a failed piece of hardware back online. Outside the context of Real Application Clusters, the acronym MTTR is also used for mean time to repair.

memory

The **storage** component used for programmatic execution and the buffering of data.

MPP

See **Massively Parallel Processing (MPP)**

MTBF

See **mean time between failures (MTBF)**

MTTF

See **mean time to failure (MTTF)**

MTTR

See **mean time to recover (MTTR)**

multi-threaded server

See **shared server**

N

See **null (N) mode**

Net8

Obsolete term. See **Oracle Net**

node

A node is a machine where an **instance** resides.

non-resilience

See **resilience**

non-uniform memory access (NUMA)

A **memory** architecture used in multiprocessors where the access time depends on the memory location. A processor can access its local memory faster than non-local memory. (Memory that is local to another processor or shared between processors.) (See also: **uniform memory access (UMA)**)

null (N) mode

A null (N) mode indicates that the holding process has an interest in a resource. However, access rights are only conferred when in **exclusive (X) mode** or **shared (S) mode**.

NUMA

See **non-uniform memory access (NUMA)**

Object-Relational Database Management System (ORDBMS)

This term is used interchangeably with **Relational Database Management System (RDBMS)**.

OCI

See **Oracle Call Interface (OCI)**

OEM

See **Oracle Enterprise Manager**

OLTP

See **Online Transaction Processing (OLTP)**

Online Transaction Processing (OLTP)

The processing of transactions by computers in real time. (See also: **transaction systems**.)

Operating System

The low-level software that handles the interface to peripheral hardware, schedules tasks, allocates storage, and presents a default interface to the user when no application program is running.

The Operating System can be split into a kernel that is always present and various system programs that use facilities provided by the kernel to perform higher-level house-keeping tasks, often acting as servers in a client/server relationship.

Operating System-Dependent (OSD) Layer

Operating System-Dependent (OSD) layers are software layers tailored for various operating systems. OSD clusterware provides communication links between the Operating System and Real Application Clusters software.

OPFS

Obsolete acronym. See **Oracle Real Application Clusters Guard**

OPSCTL

Obsolete acronym. See **SRVCTL**

Oracle9*i* Data Guard

An Oracle **high availability** product that provides a backup database on ready standby status. This product was formerly called **Standby Database**.

Oracle Call Interface (OCI)

An Application Programming Interface (API) that allows applications written in C or C++ to interact with one or more Oracle servers.

Oracle Database Configuration Assistant (DBCA)

See **Database Configuration Assistant (DBCA)**

Oracle Enterprise Manager

A system management tool that provides an integrated solution for centrally managing your heterogeneous environment. Oracle Enterprise Manager combines a graphical console, management server, Oracle Intelligent Agent, repository database, and tools to provide an integrated, comprehensive systems management platform for managing Oracle products.

A product family consists of system management tools designed to efficiently manage the complete Oracle environment.

Oracle Net

Oracle Net is a software component that enables connectivity. It includes a core communication layer called the Oracle Net foundation layer and network protocol support. Oracle Net is the foundation of Oracle's family of networking products, allowing services and their applications to reside on different computers and communicate as peer applications. The main function of Oracle Net is to establish network sessions and transfer data between a client machine and a server or between two servers. Once a network session is established, Oracle Net acts as a data courier for the client and the server. Oracle Net was formerly called Net8. (See also: **Oracle Net Services** and **Oracle Net Configuration Assistant**.)

Oracle Net Configuration Assistant

A post-installation tool that configures basic network components after installation, including:

- Listener names and protocol addresses
- Naming methods the client will use to resolve connect identifiers
- Net service names in a `tnsnames.ora` file
- Directory server access

(See also: **Oracle Net** and **Oracle Net Services**.)

Oracle Net Services

Oracle Net Services is the term that encompasses all of the Oracle networking components, including: **Oracle Net**, the **listener**, Oracle Connection Manager, Oracle Names, **Oracle Net Configuration Assistant**, and Oracle Net Manager.

Oracle parallel execution

Divides the work of processing certain types of **structured query language (SQL)** statements among multiple parallel execution server processes. (See also: **Oracle Parallel Query**.)

Oracle Parallel Fail Safe (OPFS)

Obsolete term. See **Oracle Real Application Clusters**

Oracle Parallel Query

An query coordinating option that enables you to make **structured query language (SQL)** statements parallel so that they can run on different processors in a parallel environments. When a instance is started, the Oracle database server creates a pool of query server processes. This pool is available for Oracle Parallel Query. Typically, Oracle Parallel Query is used for full table scans, sorts, subqueries, data loading, and so forth to improve performance. It is commonly used in **Decision Support System (DSS)** and data warehousing applications. (See also: **Oracle parallel execution**.)

Oracle Parallel Server

Obsolete term. See **Oracle Real Application Clusters**

Oracle Performance Manager

An add-on application for **Oracle Enterprise Manager** that offers a variety of tabular and graphic performance statistics for Real Application Clusters. The statistics represent the aggregate performance for all instances running on Real Application Clusters.

Oracle Process

See **process**

Oracle Real Application Clusters

See **Real Application Clusters**

Oracle Real Application Clusters Guard

A **failover** protection feature. Oracle Real Application Clusters Guard is an integral component of Real Application Clusters. This feature was formerly known as Oracle Parallel Fail Safe (OPFS). Oracle Real Application Clusters Guard provides the following functions:

- Automated, fast recovery and bounded recovery time from failures that crash the Oracle **instance**
- Automatic capture of diagnostic data when certain types of failures occur
- Enforced **Primary/Secondary Configuration**. Clients connecting through Oracle Net Services are properly routed to the primary node even if connected to another node in the cluster
- Elimination of delays that clients experience when reestablishing connections after a failure

Oracle System Identifier

See **system identifier (SID)**

ORDBMS

See **Object-Relational Database Management System (ORDBMS)**

OSD

See **Operating System-Dependent (OSD) Layer**

pack

A pack is a piece or suite of software that ensures the availability of the set of resources required to run an Oracle **instance**. The pack controls the startup, shutdown, and restarting of Oracle processes. There is one pack for each instance. In the context of Real Application Clusters and Oracle Real Application Clusters Guard, Packs were formerly called PFS packs.

Parallel Server

Obsolete term. See **Real Application Clusters**

parameter file (PFILE)

A file used by an Oracle server that provides specific values and configuration settings that are used at database startup. The keyword `PFILE` is used in the startup command.

password file

A file created by the `ORAPWD` command. A database must use password files if you want to connect as `SYSDBA` over a network.

past image (PI)

A past image is a copy of **dirty block** that is used by the **Global Cache Service (GCS)**. Past images of blocks are maintained until writes covering those versions are recorded. Past images are used in failure recovery.

PCTFREE

A storage parameter file that defines the percentage of space to leave in an extent.

Performance Manager

See **Oracle Performance Manager**

PFILE

See **parameter file (PFILE)**

PFS Pack

Obsolete term. See **pack**.

PI

See **past image (PI)**

ping

A synonymous term for a **forced disk write**.

planned downtime

Includes routine operations, maintenance, and upgrades that cause the system to be unavailable to users. (See also: **unplanned downtime**.)

PL/SQL

PL/SQL is Oracle's procedural extension to the structured query language (SQL). SQL is the industry-standard database access language. An advanced 4GL (fourth-generation programming language), PL/SQL offers seamless SQL access, tight integration with the Oracle server and tools, portability, security, and modern software engineering features such as data encapsulation, overloading, exception handling, and information hiding.

PMON

See **process monitor (PMON)**

preferred primary node

The node where the **pack** with the primary role resides by default at startup. (See also: **preferred secondary node**.)

preferred secondary node

The node where the **pack** with the secondary role resides by default at startup. (See also: **preferred primary node**.)

primary instance

In a primary/secondary configuration, the **instance** through which all clients access the database. (See also: **secondary instance**.)

primary instance role

In a primary/secondary configuration, the **instance** that mounts the database first assumes the primary role. It performs the work requested by application sessions. If the primary instance fails or is shut down, then **failover** occurs, and another instance assumes the primary instance role. (See also: **Primary/Secondary Configuration** and **secondary instance role**.)

Primary/Secondary Configuration

A configuration in which the primary **instance** is the instance where all clients access the database. The secondary instance provides backup services to the primary instance in case the primary instance fails. (See also: **primary instance**, **primary instance role**, **secondary instance**, and **secondary instance role**.)

private rollback segment

A rollback segment that is acquired exclusively by an **instance** when the instance opens a database.

process

An Oracle **instance** has two types of processes: User processes and Oracle processes.

- A user process executes the code of an application program (such as an Oracle Forms application) or an Oracle Tool (such as a **listener**).
- Oracle Processes are server processes that perform work for the user processes and background processes that perform maintenance work for the Oracle server.

process monitor (PMON)

A process monitor database process that performs process recovery when a user **process** fails. PMON is responsible for cleaning up the cache and freeing resources that the process was using. PMON also checks on dispatcher and server processes and restarts them if they have failed. As a part of **service registration**, PMON registers **instance** information with the **listener**.

public rollback segment

A rollback segment that is available to any **instance** that requires a rollback segment.

RAID

See **Redundant Array of Independent Disks (RAID)**

raw device

A raw device is a disk drive that does not yet have a file system set up. Raw devices are used for Real Application Clusters since they enable the sharing of disks. See also **raw partition**.

raw volumes

See **raw device**.

raw partition

A raw partition is a portion of a physical disk that is accessed at the lowest possible level. A raw partition is created when an extended partition is created and logical partitions are assigned to it without any formatting. Once formatting is complete, it is called **cooked partition**. See also **raw device**.

RDBMS

See **Relational Database Management System (RDBMS)**

reader/writer contention

In Real Application Clusters, if a session (reader) needs to read a data block that has been recently modified by another session in a different **instance** (writer), a concurrency control mechanism takes place to ensure read consistency. The **contention** resolution mechanism involves **Cache Fusion**.

Real Application Clusters

An architecture that allows multiple instances to access a shared database of datafiles. Real Application Clusters is also a software component that provides the necessary **cluster database** scripts, initialization files, and datafiles needed for the Oracle Enterprise Edition and Real Application Clusters. The Real Application Clusters product was formerly called **Oracle Parallel Server**. Also note that Real Application Clusters refers to the Oracle product name while references to the **cluster database**s are generic.

record

See row.

Recovery Manager (RMAN)

The Recovery Manager is an Oracle tool that enables you to back up, copy, restore, and recover datafiles, control files, and archived redo logs. It is included with the Oracle server and does not require separate installation. You can invoke RMAN as a command line utility from the **Operating System** prompt or use the graphical user interface-based **Oracle Enterprise Manager** Backup Manager.

redo log files

Redo files contain records of all changes made to data in the database buffer cache. Every Oracle database has a set of two or more redo log files. The set of redo log files for a database is collectively known as the database's redo log. A redo log is made up of redo entries (also called redo records). Each of these is a group of change vectors describing a single atomic change to the database. The primary function of the redo log is to record all changes made to data. Should a failure prevent modified data from being permanently written to the datafiles, the changes can be obtained from the redo log and work is never lost. Redo log files are critical in protecting a database against failures. To protect against a failure involving the redo log itself, Oracle allows a multiplexed redo log so that two or more copies of the redo log can be maintained on different disks.

Redundant Array of Independent Disks (RAID)

A hardware architecture that combines multiple hard disk drives to allow rapid access to a large volume of stored data.

Relational Database Management System (RDBMS)

A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields. Certain fields can be designated as keys. This means that searches for specific values of that field will use indexing to speed them up.

relocatable IP address

A public IP address that is started by a Oracle Real Application Clusters Guard **pack**. A relocatable IP address can be moved among nodes to ensure that it is always available. This eliminates TCP/IP timeouts for new connections after a **failover**.

repository database

A repository database is a set of tables in an Oracle database that stores data required by **Oracle Enterprise Manager**. This database is separate from the database on the nodes.

resilience

A two-node **cluster** is resilient if both nodes have instances that are **active**. If the primary node has an **instance** that is active but the instance on the secondary node is down, the cluster is in a nonresilient state.

resource

In the context of the **Global Cache Service (GCS)**, a concurrency control on data blocks. (See also: **global cache element**, **enqueue**, **lock**, **resource**, **resource affinity**, and **resource mastering**.)

resource affinity

Resource affinity is the use of **dynamic resource remastering** to move the location of the resource masters for a database file to the **instance** where block operations are most frequently occurring. This optimizes the system in situations where update transactions are being executed on one instance. When activity shifts to another instance the resource affinity will correspondingly move to the new instance. If activity is not localized, then the resource ownership is hashed to the instances.

resource mastering

The method by which the **Global Cache Service (GCS)** and **Global Enqueue Service (GES)** control **resource**s. (See also: **Global Resource Directory**.)

resource mode

A concurrency control that defines global access rights for instances in a cluster.

resource role

A concurrency control that defines whether a data block is cached in only one instance (local) or if it cached in multiple instances (global).

ring configuration

A configuration in which each node serves as a primary node and also as a secondary node for another node, forming a closed ring.

RMAN

See **Recovery Manager (RMAN)**.

rollback segments

Rollback segments are records of old values of data that were changed by each transaction (whether committed or not committed). Rollback segments are used to provide read consistency, to roll back transactions, and to recover the database. Each node typically has two rollback segments, that are identified with a naming convention of RBSthread_id_rollback_number by the `ROLLBACK_SEGMENTS` parameter in the **initsid.ora** file.

row

A synonym for **record**. One row of data in a database table, having values for one or more columns. Outside the context of this manual, the term row is also defined as one set of field values in the output of query.

row cache

The memory that stores recently accessed data for an individual record (or **row**). Row caches are used so that subsequent requests for the data held in the same row can be processed more quickly.

S

See **shared (S) mode**.

Saddr

See **session address (Saddr)**.

scalability

Scalability is the ability to add additional nodes to Real Application Clusters applications and achieve markedly improved performance.

scale up

The factor that expresses how much more work can be done in the same time period by a larger system. the factor that expresses how much more work can be done in the same time period by a larger system. (See also: **speed up**.)

SCN

See **System Change Number (SCN)**.

SCUR

See **shared current (SCUR)**.

secondary instance

In a **Primary/Secondary Configuration**, the **instance** that provides backup services to the primary instance in case the primary instance fails.

secondary instance role

In a **Primary/Secondary Configuration**, the second **instance** to mount the database assumes the secondary role. The instance with the primary role performs the work that is requested by application sessions, but selected tasks such as reporting and planned operations can be performed by the instance with the secondary instance role. (See also: **primary instance**, **primary instance role**, and **secondary instance**.)

Sequence Number Value Enqueue (SV)

An **enqueue** utilized by a session when it needs a sequence value.

Server Management (SRVM)

Server Management (SRVM) is a component of **Oracle Enterprise Manager**. It is a comprehensive and integrated system management solution for Real Application Clusters. SRVM enables you to manage multiinstance databases that run in heterogeneous environments through an open client/server architecture.

In addition to managing cluster databases, SRVM enables you to schedule jobs, perform event management, monitor performance, and obtain statistics to tune **cluster** databases. Note that SRVM is not a separate product. It is an integral part of Oracle Enterprise Manager.

service control utility

See **SRVCTL**.

service name

A service name is a logical representation of a database. This is the way a database is presented to clients. A database can be presented as multiple services and a service can be implemented as multiple database instances. The service name is a string that includes:

- the global database name

- a name comprised of the database name (`DB_NAME`)
- domain name (`DB_DOMAIN`)

The service name is entered during installation or database creation.

If you are not sure what the global database name is, you can obtain it from the combined values of the `SERVICE_NAMES` parameter in the common database initialization file, **initdbname.ora**.

The service name is included in the `CONNECT_DATA` part of the connect descriptor.

service registration

A feature whereby the **process monitor (PMON)** automatically registers information with a **listener**. Because this information is registered with the listener, the `listener.ora` file does not need to be configured with this static information.

Service registration provides the listener with the following information:

- The service names for each running **instance** of the database
- Instance names of the database
- Service handlers (dispatchers and dedicated servers) available for each instance This allows the listener to direct a client's request appropriately.
- Dispatcher, instance, and node load information. This load information allows the listener to determine which dispatcher can best handle a client connection's request. If all dispatchers are blocked, then the listener can spawn a dedicated server for the connection.

session address (Saddr)

The session address (Saddr) is the dotted notation locator for a current session. The Saddr number is displayed in both the Cluster Database Session Chart and in the `V$SESSION` dynamic performance view.

SETLINKS

SETLINKS is a utility used to map symbolic links to **raw device**s for Windows NT and Windows 2000.

SGA

See **System Global Area (SGA)**.

shared current (SCUR)

The buffer state name for a shared resource.

shared (S) mode

Shared is a protected read resource mode. No writes are allowed in shared mode. In shared mode, any number of users can have simultaneous read access to a resource. (See also: **exclusive (X) mode** and **null (N) mode**.)

shared cache

The aggregation of the buffer caches of all instances in a cluster database.

shared server

A server that is configured to allow many user processes to share very few server processes, so that the number of users that can be supported is increased. With shared server configuration, many user processes connect to a dispatcher. The dispatcher directs multiple incoming network session requests to a common queue. An idle shared server **process** from a shared pool of server processes picks up a request from the queue. This means a small pool of server processes can serve a large amount of clients. A shared server is in contrast with a dedicated server.

SID

See **system identifier (SID)**.

SMON

See **system monitor (SMON)**.

SMP

See **Symmetric Multi-Processor (SMP)**.

speed up

The concept that more hardware can perform the same task in less time than the original system. With added hardware, speed up holds the task constant and measures time savings. (See also: **scale up**.)

SQL

See **structured query language (SQL)**.

SRVCTL

Real Application Clusters can use the Service Control (SRVCTL) utility to manage instances. SRVCTL is installed on each node. The SRVCTL utility gathers information about all the instances for **Oracle Enterprise Manager**. SRVCTL acts as a single point of control between the Oracle Intelligent Agent and the nodes. Only one node's Oracle Intelligent Agent is used to communicate to SRVCTL. SRVCTL on that node then communicates to the other nodes through Java Remote Method Invocation (RMI).

SRVM

See **Server Management (SRVM)**.

Standby Database

See **Oracle9i Data Guard**.

Startup (START)

Startup is an Operating System-Dependent component that provides one-time configuration to startup functionality.

storage

A **memory** device that stores data. Usually a persistent storage that must be accessed by read / write transactions to alter its contents.

Storage Area Network (SAN)

A high speed network of shared storage devices. Typically, SAN architecture allows access to all storage devices by all servers on a local area network (LAN) or wide are network (WAN). Most SANs conform to **Fibre Channel** standards.

structured query language (SQL)

SQL is an industry-standard language for creating, updating, and querying relational database management systems. The acronym SQL is spoken "sequel".

SV

See **Sequence Number Value Enqueue (SV)**.

Symmetric Multi-Processor (SMP)

Two or more similar processors connected through a high-bandwidth link and managed by one operating system, where each processor has equal access to I/O devices. (See also: **uniform memory access (UMA)**

System Change Number (SCN)

A logical time stamp that defines a committed version of a database at one point in time. Oracle assigns every committed transaction a unique SCN.

System Global Area (SGA)

A group of shared memory structures that contain data and control information for one Oracle database **instance**. The SGA and Oracle processes constitute an Oracle instance. Oracle automatically allocates memory for an SGA whenever you start an instance and the operating system reclaims the memory when you shut down the instance. Each instance has only one SGA.

system identifier (SID)

The Oracle system identifier (SID) identifies a specific **instance** of the running Oracle software. For a Real Application Clusters database, each **node** within the **cluster** has an instance referencing the database.

The database name, specified by the `DB_NAME` parameter in the `INITDB_NAME.ORA` file, and unique **thread** ID make up each node's SID. The thread ID starts at 1 for the first instance in the cluster, and is incremented by 1 for the next instance, and so on.

system monitor (SMON)

The system monitor performs crash recovery when a failed instance starts up again. In Real Application Clusters, the SMON process of one **instance** can perform instance recovery for other instances that have failed. SMON also cleans up temporary segments that are no longer in use and recovers expired transactions that were skipped during crash and instance recovery because of file-read or offline errors. These transactions are eventually recovered by SMON when the tablespace or file is brought back online. SMON also coalesces free extents within the database's dictionary-managed tablespaces to make free space contiguous and easier to allocate.

table lock

Table locks are **Database Mount Lock**s that protect entire tables. A transaction acquires a table resource when a table is modified by one of the following statements: `INSERT`, `UPDATE`, `DELETE`, `SELECT` (with the `FOR UPDATE` clause), and `LOCK TABLE`. A table lock can be held in any of three modes: **null (N) mode**, **shared (S) mode**, and **exclusive (X) mode**.

tablespace

A logical portion of an Oracle database used to allocate **storage** for data. Each tablespace corresponds to one or more physical datafiles. Every Oracle database has a tablespace called `SYSTEM` and can have additional tablespaces. A tablespace is used to group related logical structures together. For example, tablespaces commonly group all of an application's objects to simplify certain administrative operations.

TAF

See **Transparent Application Failover (TAF)**.

takeover

Occurs when the secondary node executes **failover** of the **primary instance role** to itself. Occurs only when the primary **instance** is unavailable and the primary instance role has not resumed normal function on a new node. (See also: **primary instance**, **primary instance role**, **Primary/Secondary Configuration**, **secondary instance**, and **secondary instance role**.)

TEMP

See **temporary (TEMP) tablespace**.

temporary (TEMP) tablespace

The tablespace of temporary tables or indexes created during the processing of an **structured query language (SQL)** statement.

thread

Each Oracle **instance** has its own set of online redo log groups. These groups are called a thread of online redo. In non-Real Application Clusters environments, each database has only one thread that belongs to the instance accessing it. In Real Application Clusters environments, each instance has a separate thread, that is, each instance has its own online redo log. Each thread has its own current log member.

three nines availability

A colloquial term for 99.9% system availability.

TNS

See **Transport Network Services (TNS)**.

tnsnames.ora

A file that contains net service names. This file is needed on clients, nodes, the Console, and the Oracle Performance Manager machine.

transaction free list

A list of blocks freed by uncommitted transactions.

transaction systems

Transaction systems are application systems that are characterized by updates to the database. Examples of transaction systems include e-business systems and ERP applications. More specialized examples include telephone call and billing systems, credit card transactions, and airline reservation systems. Transaction systems are also known as **Online Transaction Processing (OLTP)** systems.

transparency

In computer software, an action is transparent if it takes place without any effect visible to the user. Transparency is considered a good system characteristic because it shields the user from the system's complexity.

Transparent Application Failover (TAF)

A runtime **failover** for **high availability** environments, such as Real Application Clusters and **Oracle Real Application Clusters Guard**. TAF refers to the failover and reestablishment of application-to-service connections. It allows client applications to automatically reconnect to the database if the connection fails, and optionally resume a `SELECT` statement that was in progress. This reconnect happens automatically from within the **Oracle Call Interface (OCI)** library.

Transport Network Services (TNS)

TNS is a foundational technology, part of Oracle Names. TNS works with any standard network transport protocol. In other contexts, TNS is also used as an acronym for Transparent Network Services.

uniform memory access (UMA)

In uniform memory access configurations, or UMA, all processors can access main memory at the same speed. In this configuration, memory access is uniform. This configuration is also known as **Symmetric Multi-Processor (SMP)**. (See also: **non-uniform memory access (NUMA)**

uniprocessor

A computer with a single CPU.

unplanned downtime

System downtime that includes system faults, data and media errors, and site outages that cause the system to be unavailable to users. (See also: **planned downtime**.)

volume manager

Software that provides storage management for a **Storage Area Network (SAN)**. Usually host-based **Redundant Array of Independent Disks (RAID)** software. Most volume managers use a graphical user interface (GUI) console to show the partitioning and status of storage volumes. Volume managers can create volumes from various disks, and those volumes can encompass multiple disks. (See also: **raw partition**.)

wake up call

See **acquisition interrupt**.

warming the library cache

The process of transferring information about parsed **structured query language (SQL)** statements and compiled **PL/SQL** units from the library cache on the primary **instance** to the library cache on the secondary instance. Warming the cache improves performance after **failover** because the library cache is already populated.

X

See **exclusive (X) mode**.

XCUR

See **exclusive current (XCUR)**.

Index