Oracle9*i* Real Application Clusters

Deployment and Performance

Release 1 (9.0.1)

July 2001

Part No. A89870-02

ORACLE

Oracle9*i* Real Application Clusters Deployment and Performance, Release 1 (9.0.1)

Part No. A89870-02

Primary Author: Mark Bauer.

Primary Contributors: David Austin, Mitch Flatland, Bill Kehoe, Kotaro Ono, Stefan Pommerenk, Jim Rawles, Joao Rimoli, and Michael Zoll.

Contributors: Lance Ashdown, Cathy Baird, Bill Bridge, Wilson Chan, Gang Chen, Carol Colrain, Sohan DeMel, Merrill Holt, John Kennedy, Raj Kumar, Tirthankar Lahiri, Neil MacNaughton, Vinay Srihari, Bob Thome, Alex Tsukerman, and Tak Wang.

Graphic Designer: Valarie Moore.

Contents

2 Online E-Commerce and Data Warehousing Application Deployment in Real Application Clusters

Part II Scaling Applications and Designing Databases for Real Application Clusters

3 Scaling Applications for Real Application Clusters

4 Database Design Techniques for Real Application Clusters

6 Tuning Real Application Clusters and Inter-Instance Performance

Part IV Using Oracle Enterprise Manager to Monitor and Tune Real Application Clusters Databases

7 Monitoring Performance with Oracle Performance Manager

B A Case Study in Real Application Clusters Database Design

List of Figures

List of Tables

Send Us Your Comments

Oracle9*i* Real Application Clusters Deployment and Performance, Release 1 (9.0.1)

Part No. A89870-02

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

Oracle9i Real Application Clusters Deployment and Performance explains the deployment issues for applications that use an Oracle9*i* Real Application Clusters database. This manual also provides post-deployment information about tuning Real Application Clusters environments for optimal performance.

Information in this manual applies to Real Application Clusters as it runs on all operating systems. Where necessary, this manual refers to platform-specific documentation.

> **See Also:** The *Oracle9i Real Application Clusters Documentation Online Roadmap* to help you use the online Oracle9*i* Real Application Clusters Documentation set

This preface contains these topics:

Audience

Oracle9i Real Application Clusters Deployment and Performance is written for database administrators and application developers working with Real Application Clusters. To use this document you should have a conceptual understanding of Real Application Clusters processing and its software and hardware components as described in *Oracle9i Real Application Clusters Concepts* and you should have installed Real Application Clusters using the document *Oracle9i Real Application Clusters Installation and Configuration* and related platform-specific documentation.

Organization

The five parts of this book and their contents are:

Part I: Introduction to Real Application Clusters Deployment

Part One introduces the high-level aspects of deploying applications in Real Application Cluster by describing how to take advantage of Oracle9*i* features. It also describes deployment of internet-based applications in e-commerce and data warehousing environments.

Chapter 1, "Introduction to Application Deployment for Real Application Clusters"

This chapter provides an overview of deployment issues for Real Application Clusters environments.

Chapter 2, "Online E-Commerce and Data Warehousing Application Deployment in Real Application Clusters"

This chapter briefly describes the deployment of online e-commerce-based and data warehousing applications for Real Application Clusters.

Part II: Scaling and Designing Applications for Oracle9*i* Real Application Clusters

Part Two describes technical issues and database design techniques for deploying scalable applications with Real Application Clusters.

Chapter 3, "Scaling Applications for Real Application Clusters"

This chapter describes how to optimize the scalability of your applications for deployment in Real Application Clusters environments.

Chapter 4, "Database Design Techniques for Real Application Clusters"

This chapter describes Real Application Clusters database design issues such as block and extent operations, contention reduction, and resource control strategies.

Part III: Oracle9*i* Real Application Clusters Performance Monitoring and Tuning

Part Three describes procedures for monitoring and tuning performance in Real Application Clusters.

Chapter 5, "General Tuning Recommendations for Real Application Clusters"

This chapter presents general tuning recommendations for Real Application Clusters.

Chapter 6, "Tuning Real Application Clusters and Inter-Instance Performance"

This chapter describes how to monitor and tune inter-instance performance in Real Application Clusters.

Part IV: Monitoring and Tuning Real Application Clusters Databases with Oracle Enterprise Manager

Part Four describes monitoring the performance of Real Application Clusters databases using Oracle Enterprise Manager.

Chapter 7, "Monitoring Performance with Oracle Performance Manager"

This chapter explains how monitor and tune Real Application Clusters databases with Oracle Enterprise Manager.

Part V: Oracle Real Application Clusters Reference

Part Five contains reference information for Real Application Clusters deployment and performance.

Appendix A, "Configuring Multi-Block Lock Assignments (Optional)"

This appendix explains how to override the Real Applications default resource control mechanisms.

Appendix B, "A Case Study in Real Application Clusters Database Design"

This appendix describes a case study for deploying Real Application Clusters.

Glossary

The glossary defines terms used in this book as well as terms relevant to the subject matter of this book.

Related Documentation

For more information, see these Oracle resources:

- *Oracle9i Real Application Clusters Documentation Online Roadmap*
- *Oracle9i Real Application Clusters Concepts*
- *Oracle9i Real Application Clusters Installation and Configuration*
- *Oracle9i Real Application Clusters Administration*
- *Oracle Real Application Clusters Guard Administration and Reference Guide*
- Your platform-specific Oracle Real Application Clusters Guard installation guide

Installation Guides

- Your platform-specific Oracle Real Application Clusters Guard installation guide
- *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*

Operating System-Specific Administrative Guides

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

Oracle9*i* Real Application Clusters Management

- *Oracle9i Real Application Clusters Administration*

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

Generic Documentation

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

Many of the examples in this book use the sample schemas of the **seed database**, which is installed by default when you install Oracle. Refer to *Oracle9i Sample Schemas* for information about how these schemas were created and how to use them.

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, please visit the Oracle Technology Network (OTN). 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 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.	When you specify this clause, you create an **index-organized table**.
Italics	Italic typeface indicates book titles or emphasis.	*Oracle9i Database Concepts* Ensure that the recovery catalog and target database do *not* reside on the same disk.
`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, usernames, and roles.	You can specify this clause only for a `NUMBER` column. You can back up the database by using the `BACKUP` command. Query the `TABLE_NAME` column in the `USER_TABLES` data dictionary view. Use the `DBMS_STATS.GENERATE_STATS` procedure.

Convention	Meaning	Example
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates executables, filenames, directory names, 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, usernames and roles, program units, and parameter values. **Note:** Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.	Enter `sqlplus` to open SQL*Plus. The password is specified in the `orapwd` file. Back up the datafiles and control files in the `/disk1/oracle/dbs` directory. 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. The `JRepUtil` class implements these methods.
lowercase monospace (fixed-width font) italic	Lowercase monospace italic font represents placeholders or variables.	You can specify the *`parallel_clause`*. Run `U`*`old_release`*`.SQL` where *`old_release`* refers to the release you installed prior to upgrading.

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]`

Convention	Meaning	Example
`...`	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , coln 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 shown.	`acctbal NUMBER(11,2);` `acct CONSTANT NUMBER(4) := 3;`
Italics	Italicized text indicates placeholders or variables for which you must supply particular values.	`CONNECT SYSTEM/system_password` `DB_NAME = database_name`

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.
	Note: Some programmatic elements use a mixture of UPPERCASE and lowercase. Enter these elements as shown.
		`SELECT last_name, employee_id FROM employees;`
		`sqlplus hr/hr`
		`CREATE USER mjones IDENTIFIED BY ty3MU9;`

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 Cluster Software Deployment and Performance?

This section describes the features of Oracle9*i* release 1 (9.0.1) as they pertain to application deployment and tuning in Real Application Clusters. This section contains the following topic:

- Oracle9i Release 1 (9.0.1) New Features for Application Deployment and Tuning in Real Application Clusters

> **See Also:** *Oracle9i Real Application Clusters Concepts* for a more complete explanation of new features for Real Application Clusters

Oracle9*i* Release 1 (9.0.1) New Features for Application Deployment and Tuning in Real Application Clusters

The Oracle9*i* release 1 (9.0.1) Real Application Clusters features and enhancements described in this section are part of an effort to simplify application deployment. Real Application Clusters is a new architecture offering scalability and high availability features that exceed the capabilities of previous Oracle cluster software releases.

- **Cache Fusion's breakthrough technology simplifies the deployment for many types of online e-commerce and decision support (DSS) applications**

  You can deploy a wide variety of application types on Real Application Clusters without significant code modifications.

  > **See Also:** Part One, "Introduction to Deployment in Real Application Clusters"

- **Cache Fusion automatically controls resource assignments for files accessed by applications in Real Application Clusters environments**

  You no longer have to assign locks to datafiles or partition your application when launching applications on Oracle cluster software. In addition, Real Application Clusters introduces direct cache-to-cache block transfers to improve performance.

  > **See Also:** Part Two, "Scaling Applications and Designing Databases for Real Application Clusters"

- **Three new views allow you to examine how Oracle is mastering 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 instances
  - `V$HVMASTER_INFO` for Global Enqueue Service resources

  These views track the current and previous master instances and the number of global cache (`V$GCSHVMASTER_INFO`) and global cache resources belonging to a file accessed frequently by a single instance (`V$GCSPFMASTER_INFO`), as well as the number of re-masterings of enqueue (`V$HVMASTER_INFO`) resources.

- **Automatic undo management**:
  - Oracle Corporation recommends that you deploy Real Application Clusters applications using automatic undo management. Automatic undo management minimizes the administrative overhead of managing undo tablespaces.

    See Also: *Oracle9i Real Application Clusters Administration* for more information on using automatic undo management

Part I

Introduction to Deployment in Real Application Clusters

Part One introduces the topic of application development for Real Application Clusters. Part One explains how to take advantage of Oracle's high availability and scalability features in Real Application Clusters environments. The chapters in Part One are:

1

Introduction to Application Deployment for Real Application Clusters

This chapter describes **Oracle Real Application Clusters** application deployment issues. This chapter first explains considerations for taking advantage of the scalability features of Real Application Clusters. It then explains some of the general issues of application deployment in Real Application Clusters environments.

This chapter includes the following topics:

Taking Full Advantage of Oracle9*i* Features

To optimally deploy applications within Oracle9*i* Real Application Clusters environments, consider the issues for the Oracle features described in this chapter. Proper implementation of these features minimizes deployment and integration problems. It also ensures that your system takes full advantage of the breakthrough technology of **Cache Fusion** and the high-performance features of Real Application Clusters.

The feature descriptions in this chapter provide a starting point for Real Application Clusters application deployment. For additional information about these features, refer to other Oracle documentation as noted.

Implementing Oracle9*i* Features with Real Application Clusters

You should consider several application deployment issues to optimize Oracle9*i* Real Application Clusters performance. These issues relate to features that are unique to Oracle and that enhance the performance of Real Application Clusters. The features discussed in this section are:

- High Availability and Failover in Real Application Clusters
- Oracle Net in Real Application Clusters
- The Shared Server in Real Application Clusters
- Connection Load Balancing
- Transparent Application Failover in Real Application Clusters
- PL/SQL in Real Application Clusters
- Recovery Manager (RMAN) in Real Application Clusters
- Cluster File Systems in Real Application Clusters

High Availability and Failover in Real Application Clusters

High availability systems are systems that have redundant hardware and software that maintains operations despite failures. Well designed high availability systems avoid having single points-of-failure. When failures occur, failover moves the processing performed by the failed component to the backup component. Oracle's failover process quickly re-masters resources, recovers partial or failed transactions, and rapidly restores the system.

You can combine many Oracle products and features to create highly reliable computing environments. Doing this requires capacity and redundancy planning. You must also set expectations and examine your service level agreements. In addition, consider your overall system costs and your return on investment. There are also other practical considerations such as selecting the appropriate hardware and deciding whether to use idle machines that simultaneously serve as part of your high availability configuration.

Primary/Secondary and Active/Active Instance Configurations

Primary / Secondary configurations are the least complicated type of high availability configuration to set up. These are also the easiest type of configurations to administer. For example, the administrative overhead for a primary database in this configuration is the same as the overhead of a single **instance** configuration.

In Primary / Secondary configurations, the second instance does not have to remain idle. For example, you can use the second instance for read-only operations.

You do not have significant scalability with Primary / Secondary configurations, but you do have high availability. Active / active instance configurations, on the other hand, are more complex to configure. In Active / Active configurations, performance is the critical factor.

Oracle Real Application Clusters Guard, which is an enhanced configuration of Real Application Clusters, offers yet another high availability solution. Real Application Clusters Guard tightly integrates Oracle's enhanced recovery features within the **cluster** framework of your platform. Real Application Clusters Guard is only available on specific UNIX configurations.

> **See Also:** *Oracle9i Real Application Clusters Concepts* for more information about Real Application Clusters and high availability and for more conceptual information about Real Application Clusters Guard

Oracle Net in Real Application Clusters

Oracle Net enables services and their applications to reside on different computers and allows them to communicate with each other. Oracle Net establishes network sessions and transfers data between clients and servers or between two servers. Real Application Clusters requires Oracle Net to enable connectivity; you must install Oracle Net on each machine in your network.

> **See Also:** The *Oracle Net Services Administrator's Guide* for more information about Oracle Net

The Shared Server in Real Application Clusters

Real Application Clusters with the functionality of the **shared server** can process thousands of concurrently connected database users. Shared server is extremely efficient at managing the connection load for many users; shared server operates similarly to the way that **transaction monitors** work.

Real Application Clusters with shared server significantly enhances the performance of applications running on two or more smaller computers. You do not need to rewrite your applications to use shared server. In fact, some applications perform better with shared server than without.

With shared server configurations, user processes connect to a **dispatcher**. The dispatcher then directs multiple incoming network session requests to a common queue. When a server process becomes available, the dispatcher connects the incoming request to the idle dispatcher. When the connection is no longer needed, the server process is available for another request. Thus, a small set of server processes can serve a large number of clients.

Connection Load Balancing

The **connection load balancing** feature automatically distributes connections among active instances. Connection load balancing does this based on the workload of the different **nodes** and instances in a cluster. You can use **connection load balancing**, in both shared server and **dedicated server** environments. Real Application Clusters and Cache Fusion combined with connection load balancing allow you to run many types of applications without application or data partitioning.

Note: You must install Oracle Net to use shared server and its **load balancing** features.

Transparent Application Failover in Real Application Clusters

The **transparent application failover (TAF)** feature automatically reconnects applications to the database if the connection fails. Because the re-connection happens automatically within the OCI library, you do not need to change the client application to use TAF.

Because most TAF functionality is implemented in the client-side network libraries (OCI), the client must use the Oracle Net OCI libraries to take advantage of TAF

functionality. Therefore, to implement TAF in Real Application Clusters, make sure you use JDBC OCI instead of PL/SQL packages.

> **Note:** Although Real Application Clusters supports both thin JDBC and JDBC OCI, TAF is only supported with JDBC OCI.

You can also use TAF in Primary/Secondary Instance configurations. If you do this, then use the `INSTANCE_ROLE` parameter in the Connect Data portion of the connect descriptor to configure explicit secondary instance connections.

> **See Also:** *Oracle9i Real Application Clusters Administration* for information about using the `INSTANCE_ROLE` parameter in Real Application Clusters and the *Oracle Net Services Administrator's Guide* for more detailed information and examples on this parameter

To use TAF, you must have a license for the **Oracle9i Enterprise Edition**. Because TAF was designed for Real Application Clusters, it is much easier to configure TAF for that environment. However, TAF is not restricted for use with Real Application Clusters environments. You can also use TAF for single instance Oracle. In addition, you can use TAF for the following system types:

- Oracle Real Application Clusters Guard
- Replicated Systems
- Data Guard

> **See Also:**
>
> - *Oracle9i Real Application Clusters Concepts* for more conceptual information on Oracle Real Application Clusters Guard and TAF in Real Application Clusters
> - *Oracle Net Services Administrator's Guide* and the *Oracle Call Interface Programmer's Guide* for more information on TAF
> - *Oracle9i Replication* for more information on Oracle replication

PL/SQL in Real Application Clusters

PL/SQL is Oracle's procedural extension of SQL. PL/SQL is an advanced fourth-generation programming language that offers features such as data

encapsulation, overloading, collection types, exception handling, and information hiding. PL/SQL also offers seamless SQL access, tight integration with the Oracle server, as well as tools, portability, and security.

> **See Also:** The *PL/SQL User's Guide and Reference* for more information about PL/SQL

Recovery Manager (RMAN) in Real Application Clusters

Recovery Manager (RMAN) is an Oracle tool that you can use to backup, copy, restore, and recover **datafiles**, **control files**, and archived redo logs. You can invoke RMAN as a command line utility or use it through the **Oracle Enterprise Manager**.

RMAN automates many backup and recovery tasks. For example, RMAN automatically locates the appropriate backups for each datafile and copies them to the correct destinations. This eliminates the manual, error-prone effort of using operating system commands to accomplish the same task.

You must accurately configure RMAN so that all instances can access all the archive logs throughout the cluster. When one instance fails, the surviving instance that performs recovery must access the archive logs of the failed instance.

> **See Also:** *Oracle9i Real Application Clusters Administration* for details on configuring RMAN for use with Real Application Clusters and *Oracle9i Recovery Manager User's Guide and Reference* for detailed information on RMAN

Cluster File Systems in Real Application Clusters

Oracle9*i* supports cluster file systems on a limited number of platforms. Cluster file systems simplify Real Application Clusters installation and management. Using cluster file systems eliminates the need to manage **raw devices**. Cluster file systems also offers scalable, low latency, highly resilient file systems that significantly reduce storage costs. For details on how to implement cluster file systems, refer to your vendor cluster file system documentation.

Deployment Phases for Real Application Clusters

The next chapter in Part One examines application development for online e-commerce and decision support systems. The remainder of this book examines the design and performance phases of deployment for Real Application Clusters.

1. Design your application considering the issues described in Chapter 3, "Scaling Applications for Real Application Clusters" and Chapter 4, "Database Design Techniques for Real Application Clusters".
2. Install your application, populate your database, and launch the application.
3. Monitor your Real Application Clusters environment, examine performance, and tune your Real Application Clusters database and application as described in and Part III and Part IV.

2

Online E-Commerce and Data Warehousing Application Deployment in Real Application Clusters

This chapter discusses the deployment of online e-commerce (OLTP) and data warehousing applications in **Oracle Real Application Clusters** environments. This chapter also briefly describes application performance tuning.

This chapter includes the following topics:

Cache Fusion and E-Commerce Applications for Real Application Clusters

Cache Fusion processing makes the Real Application Clusters database the optimal deployment server for online e-commerce applications. This is because these types of applications require:

- High availability
- Scalability
- Load balancing according to demand fluctuations

To accommodate high availability, Cache Fusion offers multi-instance processing capabilities to re-distribute workloads to surviving instances without interrupting processing. Real Application Clusters also automatically re-masters resources to give the **instance** with the most activity on a particular datafile, for example, a greater share of control over the resources for that **datafile**.

Cache Fusion also provides excellent scalability so that if you add or replace **node**s, Oracle automatically re-masters resources and evenly distributes processing loads without reconfiguration or application re-partitioning. Real Application Clusters also takes advantage of Oracle **load balancing** features.

Real Application Clusters' application workload management is dynamic. Real Application Clusters can alter workloads in real-time based on changing business requirements. This occurs in a manageable environment with minimal administrative overhead. The dynamic resource allocation capabilities of the Cache Fusion architecture provide optimal performance for online applications with great deployment flexibility.

Flexible Implementation with Cache Fusion

E-commerce requirements, especially the requirements of online transaction processing systems, have workloads that change frequently. To accommodate this, Real Application Clusters deployments remain flexible and dynamic. They provide a wide range of service levels that, for example, might fluctuate due to:

- Varying user demands
- Peak scalability issues like trading storms (bursts of high volumes of transactions)
- Varying availability of system resources

To address these requirements, it is impractical and often too complex to partition packaged e-commerce applications, due to the limited access to table partitioning. Once deployed, such applications can access hundreds, or even thousands, of tables. Moreover, application partitioning is a difficult if not impossible task. A common recommendation is that you use a larger server or that you segment the application or application modules across distinct databases.

Segmenting applications, however, can fragment data and constrain a global enterprise-wide view of your information. Cache Fusion eliminates such requirements by dynamically migrating database resources to adapt to changing business requirements and workloads.

Cache Fusion dynamically allocates database resources to nodes based on data access patterns. This makes the data available on demand. In other words, if an instance recently accessed the data, the data is always in a local cache. The Cache Fusion architecture also migrates resources to accelerate data access, thus making Real Application Clusters a high performance e-commerce database platform.

Because re-partitioning efforts cannot keep pace with the rapid changes in system demand, Real Application Clusters also takes advantage of other features such as the **shared server**. Shared server greatly enhances scalability by providing connection pooling and Connection Load Balancing.

Deployment Strategies for Real Application Clusters-Based Applications

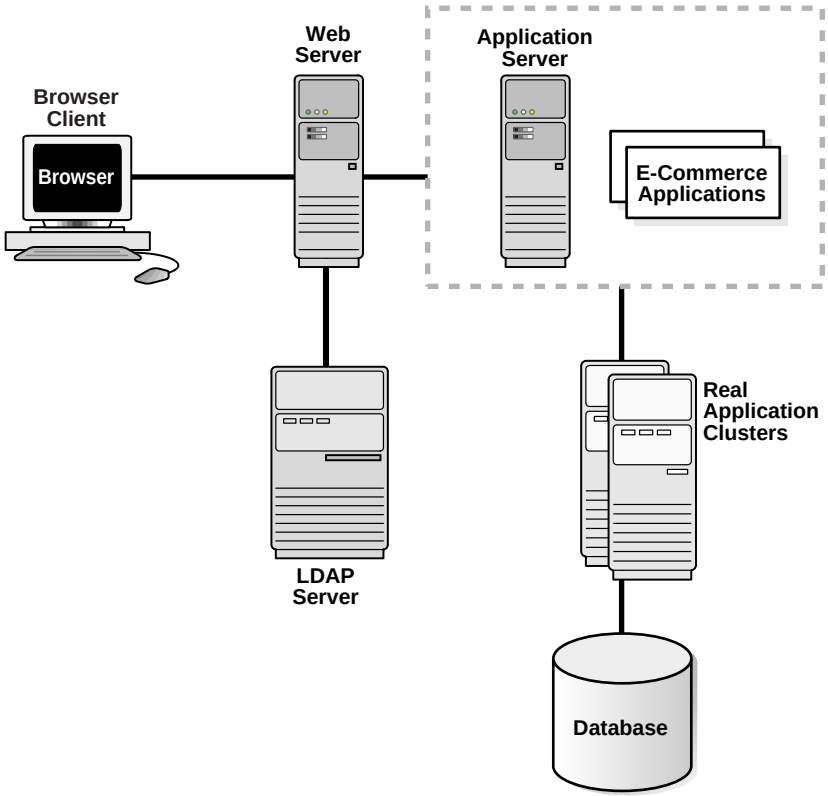
Tightly integrated, two-, three-, and *n*-tier architectures are rapidly replacing traditional, distributed designs. These architectures use a component-based approach to systems deployment. This means that in addition to the design and front-end development tasks required to launch viable web-based applications for Real Application Clusters, you may also need to consider your middleware performance requirements.

Transition to N-tier Architectures

Expensive servers residing in multiple locations are being rapidly replaced by individual servers connected to middleware components. Newer n-tier architectures are obviously less costly because they eliminate redundant server hardware. They also eliminate redundant database connections and reduce the amount of data that must travel through the network.

In addition, traditional architectures are transitioning from requiring static, inflexible connections to using full-time, internet-based connectivity. This is especially true for business-to-business (B2B) models. Real Application Clusters allows seamless integration into the n-tier model's component-based architecture as shown in Figure 2–1.

Figure 2–1 N-tier Architecture Middleware Components and Real Application Clusters



Benefits of N-Tier Architectures with Real Application Clusters

N-tier architectures provide enhanced scalability by encapsulating application functions within smaller subcomponents. These components are then linked within the n-tier framework. This combination:

- Decreases the network load on any one part of the infrastructure
- Minimizes or eliminates single-points-of-failure
- Significantly reduces network traffic

N-tier architectures also improve manageability by reducing hardware and software overhead. In addition, n-tier architectures offer high availability and increased reliability by providing connection pooling and enhanced load balancing.

Monitoring and Tuning Performance in N-Tier Environments

Because n-tier environments have more interrelated, interdependent components than traditional models, you should monitor and tune the performance of your Real Application Clusters environment at the following levels:

- Application
- Database
- Operating System

> **See Also:** The *Oracle9i Database Performance Guide and Reference* and the *Oracle9i Data Warehousing Guide* for information that is specific to tuning the Oracle database, and your vendor's documentation for operating system tuning information

Deploying Data Warehousing Applications for Real Application Clusters

This section discusses deploying data warehousing systems in Real Application Clusters environments by briefly describing the data warehousing features available in shared disk architectures. The topics in this section are:

- Speed-Up for Data Warehousing Applications on Real Application Clusters
- Dynamic Parallel-Aware Query Optimization

> **See Also:** *Oracle9i Data Warehousing Guide* for detailed information about implementing data warehousing in Real Application Clusters environments

Speed-Up for Data Warehousing Applications on Real Application Clusters

E-businesses strategically use data warehousing systems to acquire customers and expand markets. For example, company product promotions gather information using data warehousing systems to customize client lists that best fit the profiles of the company's target demographics.

Real Application Clusters is ideal for data warehousing applications because it augments the single instance benefits of Oracle. Real Application Clusters does this by maximizing the processing available on all nodes of a **cluster** to provide speed-up and scale-up for data warehousing systems.

Flexible Parallelism within Real Application Clusters Environments

Oracle's **parallel execution** feature uses multiple processes to execute SQL statements on one or more CPUs. Parallel execution is available on both single instance and Real Application Clusters databases.

Real Application Clusters takes full advantage of parallel execution by distributing parallel processing to all the nodes in your cluster. The number of processes that can participate in parallel operations depends on the **degree of parallelism (DOP)** assigned to each table or index.

Function Shipping On loosely coupled systems, Oracle's parallel execution technology uses a function shipping strategy to perform work on remote nodes. Oracle's parallel architecture uses function shipping when the target data is located on the remote node. This delivers efficient parallel execution and eliminates unneeded inter-node data transfers over the **interconnect**.

Exploitation of Data Locality On some hardware systems, powerful data locality capabilities were more relevant when shared nothing hardware systems were popular. However, almost all current cluster systems use a shared disk architecture.

On shared nothing systems, each node has direct hardware connectivity to a subset of disk devices. On these systems it is more efficient to access local devices from the *owning* nodes. Real Application Clusters exploits this affinity of devices to nodes and delivers performance that is superior to shared nothing systems using multi-computer configurations and a shared disk architecture.

As with other elements of Cache Fusion, Oracle's strategy works transparently without data partitioning. Oracle dynamically detects the disk on which the target data resides and makes intelligent use of the data's location in the following two ways:

- Oracle spawns parallel execution server processes on nodes where the data to be processed is located
- Oracle assigns local data partitions to each sub-process to eliminate or minimize inter-node data movement

Dynamic Parallel-Aware Query Optimization

Oracle's cost-based optimizer considers parallel execution when determining the optimal execution plans. The optimizer dynamically computes intelligent heuristic defaults for parallelism based on the number of processors.

An evaluation of the costs of alternative access paths—table scans versus indexed access, for example—takes into account the degree of parallelism available for the operation. This results in Oracle selecting execution plans that are optimized for parallel execution.

Oracle also makes intelligent decisions in Real Application Clusters environments, with regard to intra-node and inter-node parallelism. In intra-node parallelism, for example, if a SQL statement requires six query sub-processes and six CPUs are idle on the local node (the node to which the user is connected), the SQL statement is processed using local resources. This eliminates query coordination overhead across multiple nodes.

Continuing with this example: if there are only two CPUs on the local node, then those two CPUs and four CPUs of another node are used to complete the SQL statement. In this manner, Oracle uses both inter-node and intra-node parallelism to provide speed-up for query operations.

In all real world data warehousing applications, SQL statements are not perfectly partitioned across the different parallel execution servers. Therefore, some CPUs in the system complete the assigned work and become idle sooner than others. Oracle's parallel execution technology is able to dynamically detect idle CPUs and assign work to these idle CPUs from the execution queue of the CPUs with greater workloads. In this way, Oracle efficiently re-distributes the query workload across all of the CPUs in the system. Real Application Clusters extends these efficiencies to clusters by re-distributing the work across all the nodes of the cluster.

Load Balancing for Multiple Concurrent Parallel Operations

Load balancing distributes parallel execution server processes to spread CPU and memory use evenly among nodes. It also minimizes communication and remote I/O among nodes. Oracle does this by allocating servers to the nodes that are running the fewest number of processes.

The load balancing algorithm maintains an even load across all nodes. For example, if a DOP of 8 is requested on an 8-node system with 1 CPU for each node, the algorithm places 2 servers on each node. If the entire parallel execution server group fits on one node, the load balancing algorithm places all the processes on a single node to avoid communications overhead. For example, if you use a DOP of 8 on a 2-node cluster with 16 CPUs for each node, the algorithm places all 16 parallel execution server processes on one node.

Using Parallel Instance Groups

You can control which instances allocate parallel execution server processes with **instance groups**. To do this, assign each active instance to at least one or more instance groups. Then dynamically control which instances spawn parallel processes by activating a particular group of instances.

Establish instance group membership on an instance-by-instance basis by setting the `INSTANCE_GROUPS` initialization parameter to a name representing one or more instance groups. For example, on a 32-node system owned by both a Marketing and a Sales organization, you could assign half the nodes to one organization and the other half to the other organization using instance group names. To do this, assign nodes 1-16 to the Marketing organization using the following parameter syntax in your **initialization parameter file**:

```
sid|1-16|.INSTANCE_GROUPS=marketing
```

Then assign nodes 17-32 to Sales using the following syntax in the parameter file:

```
sid|17-32|.INSTANCE_GROUPS=sales
```

Activate the nodes owned by Sales to spawn a parallel execution server process by entering the following:

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = 'sales';
```

In response, Oracle allocates parallel execution server processes to nodes 17-32. The default value for PARALLEL_INSTANCE_GROUP is all active instances.

Note: An instance can belong to one or more groups. You can enter multiple instance group names with the INSTANCE_GROUPS parameter using a comma as a separator.

Disk Affinity

Disk affinity refers to the relationship of an instance to the data that it accesses. The more often an instance accesses a particular set of data, the greater the affinity that instance has to the disk on which the data resides.

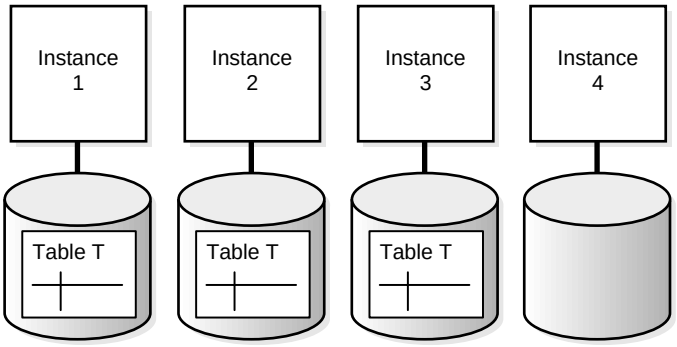
Disk affinity minimizes data shipping and internode communication on a shared nothing architecture. Disk affinity can thus significantly increase parallel operation throughput and decrease response time.

Disk affinity is used for parallel table scans, parallel temporary **tablespace** allocation, parallel DML, and parallel index scans. It is not used for parallel table creation or parallel index creation. Access to temporary tablespaces preferentially uses local datafiles. It guarantees optimal space management extent allocation. Disks striped by the operating system are treated by **disk affinity** as a single unit.

Without disk affinity, Oracle attempts to balance the allocation of parallel execution servers evenly across instances. With disk affinity, Oracle allocates parallel execution servers for parallel table scans on the instances that are closest to the requested data.

In the disk affinity example in Figure 2–2, table T is distributed across 3 nodes, and a full table scan on table T is being performed.

Figure 2–2 Disk Affinity Example



- If a query requires 2 instances, then two instances from the set 1, 2, and 3 are used.
- If a query requires 3 instances, then instances 1, 2, and 3 are used.
- If a query requires 4 instances, then all four instances are used.
- If there are two concurrent operations against table T, each requiring 3 instances (and enough processes are available on the instances for both operations), then both operations use instances 1, 2, and 3. Instance 4 is not used. In contrast, without disk affinity, instance 4 is used.

> **See Also:** *Oracle9i Real Application Clusters Concepts* for more information on instance affinity

Deployment and Tuning of Real Application Clusters Applications

Testing application scalability, availability, and load balancing is one of the most challenging aspects of internet-based deployment. In rapid prototyping environments, you can internally test and benchmark your site with a limited number of Real Application Clusters instances on a test hardware platform.

All applications have limits on the number of users they support given the constraints of specific hardware and software architectures. To accommodate

anticipated demand, you can estimate the traffic loads on your system and determine how many nodes you need. You should also consider peak workloads.

Your scalability tests must also simulate user access to your web site. To do this, configure your traffic generator to issue pseudo `get` commands as used in the Hypertext Transfer Protocol (http). This tests your system's performance and load processing capabilities under fairly realistic conditions.

You can then conduct structured, web-based testing using traffic generators to stress-test your system. This type of persistent testing against your most aggressive performance goals should reveal design and capacity limitations.

After making further enhancements as dictated by your testing results, prototype your site to early adopters. This enables you to obtain real-world benchmarks against which you can further tune your system's performance.

Configuring and Tuning Applications on Real Application Clusters

As mentioned in Chapter 1, Real Application Clusters takes advantage of several features that enhance scalability. For example, the shared server allows more users to access your system and efficiently controls database connectivity among users.

Optimizing application performance requires that you develop a representative workload profile using a tool that computes each application module's CPU and Program Global Area (PGA) memory usage. Oracle also records statistics about programs as well as information about how program modules connect to the database. The recommendations for developing workload profiles and tuning applications for Real Application Clusters appear in Chapter 3, "Scaling Applications for Real Application Clusters".

Administrative Aspects of System Scaling for Real Application Clusters

You may need to add nodes before deployment or during production to accommodate growth requirements or to replace failed hardware. Adding a node to your Real Application Clusters environment involves two main steps:

- Adding a node at the clusterware layer
- Adding a node at the Oracle layer

To add a node, connect the hardware according to your vendor's installation instructions. Then install the clusterware and the operating system-dependent layer (OSD). Complete the installation of the instance on the new node using the Oracle Universal Installer (OUI) and the Oracle Database Configuration Assistant (DBCA).

See Also: *Oracle9i Real Application Clusters Administration* for detailed procedures on adding nodes and instances

The next part of this book describes application scaling and database design for Real Application Clusters.

Part II

Scaling Applications and Designing Databases for Real Application Clusters

Part Two describes the issues for scaling applications and designing databases for Real Application Clusters. Part Two includes the following chapters:

- Chapter 3, "Scaling Applications for Real Application Clusters"
- Chapter 4, "Database Design Techniques for Real Application Clusters"

3

Scaling Applications for Real Application Clusters

This chapter describes methods for scaling applications for deployment in **Oracle Real Application Clusters** environments. This chapter provides a methodical approach to application design as well as procedures for resolving application performance issues. Topics in this chapter include:

- Overview of Development Techniques in Real Application Clusters
- SQL Statement Execution in Real Application Clusters
- Workload Distribution Concepts in Real Application Clusters
- Workload Characterization in Real Application Clusters
- Scaling-Up and Partitioning in Real Application Clusters

Overview of Development Techniques in Real Application Clusters

In general, application deployment for **Oracle Real Application Clusters** should not be significantly different than application deployment for single **instance** environments. There are, however, special topics and particular techniques that you should keep in mind. This chapter explains these issues and provides a high-level development methodology for deploying Real Application Clusters-based applications to optimize Oracle9*i* features.

Begin with an Analysis

To use Real Application Clusters to improve overall database throughput, conduct a detailed analysis of your database design and your application's workload. This ensures that you fully exploit the added processing power provided by the additional **node**s. Even if you are using Real Application Clusters only for high availability, careful analysis enables you to more accurately predict your system resource requirements.

A primary characteristic of high performance Real Application Clusters systems is that they minimize the computing resources used for **Cache Fusion** processing. That is, these systems minimize the number of inter-instance resource operations. Before beginning your analysis, however, you must understand how Real Application Clusters accesses database blocks when processing SQL statements that are issued by your applications. This is described in the following section.

SQL Statement Execution in Real Application Clusters

Most transactions involve a mixture of INSERT, UPDATE, DELETE, and SELECT statements. Exactly what percentage of these statement types that each transaction uses depends on the transaction type. Likewise, each of these operations accesses certain types of database blocks. You can categorize these block types as:

- Data blocks
- Index blocks (root, branch, and leaf)
- Segment header blocks
- Rollback segment header blocks
- Rollback segment blocks

Block access modes control concurrent access to database blocks in a cache. In the buffer cache, a block can be accessed in any of the following modes:

- X, or *exclusive current read* (XCUR)
- S, or *shared current read* (SCUR)
- N, or *null*, or *consistent read* (CR)

The **Global Cache Service (GCS)** maintains block access modes. To see a buffer's state as well as information about each buffer header, query the STATUS column of the V$BH dynamic performance view.

> **See Also:** *Oracle9i Real Application Clusters Concepts* for a detailed description of how the Global Cache Service performs data access

Block Accesses During INSERT Statement Execution

When Oracle executes INSERT statements, Oracle reads the segment header of a database object. This might mean that an INSERT statement must read a segment free list in the segment header of a table or an index to locate a block with sufficient space into which to fit a new row.

Therefore, to process inserts, Oracle reads the current, or most up-to-date version of the header block. If there is enough free space in the block after completing the insert, then the block remains on the free list and the transaction reads the corresponding data block and writes to it. For this sequence of events:

- The segment header is acquired in SCUR mode, which means that the **instance** must request a global S mode

- The data block is acquired in XCUR mode, or globally in X mode

If the remaining free space in the block is insufficient after the insert operation, then Oracle removes the block from the free list. This means Oracle updates the segment header block containing the free list. For this sequence of events:

1. The segment header block is first acquired in SCUR mode (global S mode).
2. After checking the block, Oracle escalates the buffer access mode to XCUR, (global X).
3. Oracle removes the block from the free list.
4. If a new block beyond the current **high water mark** is used, then Oracle raises the high water mark.
5. The data block is read in XCUR mode and written to disk.

This scenario assumes that freelist groups were not defined for the segment. In that case, Oracle stores the high water mark as well as a map of allocated extents in the segment header.

If Oracle allocates an additional extent to insert into an object, then Oracle raises the **high water mark** and updates the extent map. In other words, Oracle changes the segment header block in a consistent fashion; this also requires that Oracle lock the header block in exclusive mode.

Note: For the preceding explanations and the following descriptions, assume Oracle has cached all the blocks required for the operation in memory.

For an insert into a table with an index, even more data block accesses are required. First, Oracle reads the header block of the index segment in SCUR mode, then Oracle reads the root and branch blocks in SCUR mode. Finally, Oracle reads the leaf block in XCUR mode.

Depending on the height of the index tree, Oracle would also have to read more branch blocks. If a free list modification is required, then Oracle escalates the index segment header access mode to XCUR. If there is concurrency for the segment header due to free list modifications, then Oracle sends the header block back and forth between multiple instances. Using free list groups at table creation effectively achieves free list partitioning.

See Also: "Using Free List Groups For Concurrent Inserts from Multiple Nodes" on page 4-3

Block Accesses During UPDATE Statement Execution

An UPDATE statement always acquires a database block in its current version and sets the buffer to XCUR mode. Globally, this maps to a request for the block in S mode. Assuming all blocks are cached, an UPDATE transaction:

1. Reads the buffer in XCUR mode and gets the block in X mode.
2. Writes to the buffer and modifies a row.
3. If the updated row fits into the same block, then the instance does not need to acquire new blocks from the free list and the modification is complete; segment header access is unnecessary.
4. The instance retains exclusive access and the GCS sends the block to the requesting instance.
5. The local instance releases the resource or retains it in NULL mode. The local instance also requests the block in exclusive mode for subsequent updates; this can involve other Cache Fusion block transfers.

If the table has an index, then the UPDATE statement:

1. Reads the root block of the index in SCUR mode.
2. Reads one or more branch blocks in SCUR mode.
3. Reads the leaf block and pins it into the cache in SCUR mode.
4. Reads the data block in XCUR mode.
5. Modifies the data block.

If the index key value was changed, then Oracle:

1. Re-reads the root and branch blocks in SCUR mode.
2. Reads the leaf block in XCUR mode.
3. Modifies the index key value for the updated row. This can involve deleting the key value from the current block and acquiring another leaf block in XCUR mode.

During the update operation with an index, a block can be transferred out of the cache of the updating instance at any time and would have to be reacquired. The shared global resources on the root and branch blocks are not a performance issue,

as long as another instance reads only these blocks. If a branch block has to be modified because a leaf block splits, then Oracle escalates the S mode to an X mode, thus increasing the probability of conflict. The dispersion of key values in an index can be very important. With monotonically increasing index keys, a hot spot can be created in the right edge of the index key.

See Also: "Index Issues for Real Application Clusters Design" on page 4-16

Block Accesses During DELETE Statement Execution

Oracle accesses blocks in the cache for a delete in a similar way that it does for an update. Oracle scans the table for the block containing the row to be deleted. Therefore, if the table does not have an index, the transaction reads the segment header, reads the block, and then modifies the block. The transaction can create free space in the block so that if the data in the block drops below PCTUSED, the block is linked to the free list.

Consequently, the transaction acquires the segment header or free list group block in exclusive mode. Then the block in question is returned to the instance's free list group, if there is one. To avoid excess overhead during peak processing periods, you should schedule massive deletions to occur during off-peak hours.

Block Accesses During SELECT Statement Execution

A SELECT statement reads a buffer in either SCUR or CR mode. To access a buffer in SCUR mode, such as for segment headers, a global shared resource must be acquired. Most of the buffer accesses for SELECT statements are in CR mode and may not involve global resource operations. When a transaction needs to acquire a buffer in CR mode, three scenarios are possible:

- If another instance holds the needed block in exclusive mode, then the holding instance creates a consistent read version of the block and ships the block to the requesting instance. No resource operations are involved.
- If another instance holds the needed block in shared mode, then the holding instance transfers a copy of the block to the requesting instance; the requesting instance is granted a shared mode on the block.
- If no other instance has the block in its cache, then the requesting instance is granted a shared mode on the block and the instance reads it from disk.

For tables that have read-only data, you can greatly minimize the overhead for SELECT statements by putting read-only tables into read-only **tablespaces**. Do this

using the ALTER TABLESPACE READ ONLY statement. Making a tablespace read-only has two main advantages:

- Recovery occurs more quickly
- You only need to backup the tablespace once

> **See Also:** Appendix A, "Configuring Multi-Block Lock Assignments (Optional)"

Workload Distribution Concepts in Real Application Clusters

Partitioning distributes workloads among existing Real Application Clusters instances to effectively use hardware resources. With Cache Fusion in Real Application Clusters, partitioning is less critical. This is because the default cache coherency mechanism consumes fewer machine resources than the I/O-based forced disk write architecture used in previous Oracle releases.

To reduce Real Application Clusters overhead, each instance in a **cluster** should ideally perform most DML operations against a set of database tables that is not frequently modified by other instances. However, variables such as CPU and memory use are also important factors. In many cases, the performance and scalability gains of distributing data access based on load surpasses the loss of performance due to excess inter-instance communication.

There are no strict rules about implementing application partitioning. In general, however, you can apply several strategies to partition application workloads. These strategies are not necessarily mutually exclusive and are discussed in the following sections:

- Functional Partitioning
- Separating E-Commerce and Data Warehousing Processing
- Departmental and User Partitioning
- Physical Table Partitioning
- Transaction Partitioning

Functional Partitioning

Functional partitioning is often the first logical approach to achieve an optimally performing environment in terms of Real Application Clusters overhead. Modules and functional areas usually share only a small subset of Oracle objects, so contention is limited.

On the other hand, as you integrate all modules of your application, there will always be common objects for a given set of modules on any workload. In other words, it is impossible to completely eliminate Real Application Clusters overhead. Therefore, the ideal partitioning strategy depends on how the modules interact, as well as on how each module uses system resources.

Separating E-Commerce and Data Warehousing Processing

Another application partitioning method is to separate online e-commerce processing from data warehousing workloads. For example, by executing long running reports on one node, you can reduce excessive CPU use on another node dedicated to OLTP. This improves overall response times for OLTP users while providing more CPU power for reporting.

Although some Real Application Clusters overhead is expected when the reports read data recently modified by OLTP transactions, it is very unlikely that the overhead substantially affects performance. Reports require consistent read versions of buffers modified by the OLTP instance. To accommodate the data warehousing instance's requests, the OLTP instance constructs the consistent read buffers and transfers them to the data warehousing instance. In this case, there are minimal resource operations.

Departmental and User Partitioning

An alternative partitioning method that increases scalability is departmental and user partitioning. There are several methods for implementing departmental and user partitioning.

For one type of departmental partitioning, separate the tables by access groups based on geographic location. For example, assume a hotel reservation system processes room requests for hotels worldwide. In this case, you might partition the application by geographic markets such as:

- European Market
- North American Market
- Central and South American Market
- Asia Pacific Market

Physical Table Partitioning

By accurately implementing table partitioning by range, you can reduce concurrent access to the same blocks of a particular table from different instances. An example of employing table partitioning to reduce overhead is the way Oracle Applications 11*i* implements batch job processing.

One of the main tables used for batch processing in Oracle Applications is FND_CONCURRENT_REQUESTS, which is a batch queue table. When a user requests a batch job, Oracle inserts a row into the queue. The Concurrent Managers are processes that periodically query the queue, pick up requests to be run, and update the requests' statuses.

In a Real Application Clusters configuration with Concurrent Managers processing batch requests from two instances, there can be a high volume of DML statements on FND_CONCURRENT_REQUESTS. The DML statements might consist of INSERT statements from the requesting users and UPDATE statements from the Concurrent Managers. This can increase the cache transfer frequencies for blocks belonging to that table and its indexes.

To reduce Real Application Clusters overhead, partition the FND_CONCURRENT_REQUESTS based on the INSTANCE_NUMBER column. When a user submits a request, Oracle reads INSTANCE_NUMBER from V$INSTANCE and stores its value with other request information. That way, Oracle places requests generated from users connected to different instances on different partitions.

By default, each Concurrent Manager only processes requests from the instance to which it is connected, which means a single table partition. This nearly eliminates contention on FND_CONCURRENT_REQUESTS and its indexes, most of which were created as local indexes.

Similar table partitioning techniques can be very effective at reducing Real Application Clusters overhead for tables subject to very high volumes of DML activity. However, carefully consider the development costs associated with application changes needed to implement that type of solution. With Cache Fusion, most applications can achieve acceptable scalability without code changes.

> **See Also:** *Oracle9i Database Administrator's Guide* for more information on creating and managing partitioned tables

Transaction Partitioning

Transaction partitioning is the lowest level partitioning method. This method requires a three-tiered architecture where clients and servers are separated by a

transaction monitor processing layer. Based on the content of a transaction, the transaction monitor routes transactions that act on specific tables by way of specific nodes. The correct node for execution of the transaction is a function of the actual data values being used in the transaction. This process is more commonly known as **data dependent routing**.

Using this method, you can create and load your tables using any method because the transaction monitor determines which node processes a particular transaction. Transaction partitioning also enables you to achieve fine-grained transaction control. This makes transaction processing monitors very scalable. However, significant development effort is required to deploy this method.

You can accomplish data-dependent routing in one of two ways. If the partitioning of the tables fits well within actual partition usage patterns, in other words, you partitioned the table by locations and users are similarly partitionable, then you can accomplish manual routing by having users connect to the instance that is running the relevant application. Otherwise, the administration of data-dependent routing can be complex and can involve additional application code.

You can simplify the process if the application uses a transaction monitor or remote procedure call (RPC) mechanism. It is possible to place code into the transaction monitor's configuration that defines a data-dependent routing strategy. You must base this code on the input RPC arguments. Similarly, you could code this strategy within the procedural code using case statements to determine which instance should execute a particular transaction.

Workload Characterization in Real Application Clusters

One of the most important steps in developing scalable systems is to perform workload characterization studies. By understanding the application load characteristics, you can properly plan for growth and make use of the system resources available to provide optimal performance and scalability.

An accurate workload characterization can help you decide how to partition your application for Real Application Clusters. The steps discussed in this section describe a methodology for workload characterization to implement functional partitioning:

- Step 1: Define Your System's Major Functional Areas
- Step 2: Estimate Each Functional Area's System Resource Consumption
- Step 3: Analyze Each Functional Area's Data Access Pattern

Step 1: Define Your System's Major Functional Areas

Identify the major functions of your application. For example, assume a major hotel chain develops a system to automate the following high-level functions:

- Reservations
- Property Management and Maintenance
- Sales and Marketing
- Front Desk, Concierge, and Dining Facilities Management

Also determine which users are going to access the data from each of the functional areas.

Step 2: Estimate Each Functional Area's System Resource Consumption

It is important to estimate how much system resources, such as CPU, memory, and so on, that each module or functional area is expected to consume during peak system use. If the system is not yet in production, then this involves predicting the behavior of hypothetical workloads. Often, this estimation is not very precise.

If your system is already in production, or if you have a test system with similar characteristics, then it is easier to compute key performance indicators by module or functional area. For that, you should have an easy way to determine what application module a given database session is running. The `PROGRAM` column in the `V$SESSION` view gives you the name of the executable running on the client

side. If different modules run the same executable, then that information is not adequate. In these cases, use Oracle's DBMS_APPLICATION_INFO package to provide additional information.

For instance, Oracle Applications 11*i* calls the DBMS_APPLICATION_INFO.SET_MODULE procedure to register the Oracle Forms and Oracle Reports names in the MODULE column in V$SESSION. This is useful because you can gather all relevant session statistics from V$SESSTAT, join them to V$SESSION and group them by MODULE. Then you can break down instance statistics by module to produce a workload profile. The following example syntax does this:

```
SELECT s.module, SUM(st.value)
FROM V$SESSION S, V$SESSTAT ST
WHERE s.sid=st.sid
AND st.statistic#=12 /* CPU used by this session */
GROUP BY s.module
```

Another Oracle workload characterization feature is the BEFORE LOGOFF ON DATABASE event trigger. Use this trigger with the information from V$SESSION to gather relevant statistics from sessions just before they disconnect and store those statistics in a table. Because this operation requires only one INSERT … AS SELECT statement for each session, the associated overhead can be minimal. However, take care not to gather too many unnecessary statistics that might increase the trigger overhead. With that in mind, you can use this trigger in a production environment with minimal effect on performance. The following syntax is an example of this:

```
CREATE OR REPLACE TRIGGER my_logoff
BEFORE LOGOFF ON DATABASE
BEGIN
 INSERT INTO w_session
  (instance_number, logoff_time, sid, audsid, module, program, cpu, pgamem)
  SELECT p.value,
SYSDATE,s1.sid,s1.audsid,s1.module,s1.program,s2.value,s3.value
  FROM V$SESSION s1, V$MYSTAT s2, V$MYSTAT s3, V$PARAMETER p
  WHERE s2.statistic#=12  /* CPU used by this session */
and s3.statistic#=21 /* session pga memory max */
AND s1.sid=s2.sid
AND s1.sid=s3.sid
AND p.name='instance_number';
END;
/
```

After computing CPU and memory use for each module during peak system use, the possibilities in terms of application partitioning become clearer and solutions

also become more obvious. With this type of information, you can distribute application modules among existing instances so that you can fully exploit the capacity of each node.

However, you should not base your partitioning strategy only on system resource consumption. In some cases, in high volume OLTP systems with different application modules frequently modifying the same tables, Real Application Clusters overhead can become an important factor for determining the ideal scaling configuration. In this case, also consider how each functional area accesses data.

> **See Also:** *Oracle9i Supplied PL/SQL Packages Reference* for more information about Oracle packages

Step 3: Analyze Each Functional Area's Data Access Pattern

Functional areas that access disjoint sets of tables perform best with Real Application Clusters. For that reason, your focus in data access analysis should be to identify and study tables that are accessed by more than one functional area—in other words, overlaps among functional areas.

Step 3.1: Identify Table Access Requirements and Define Overlaps

Determine which tables each functional area accesses and identify the overlaps. Overlaps are tables that users from more than one functional area access. Table 3–1 shows the overlapping tables from this example in bold; the remaining tables are accessed exclusively by the functions denoted by the column headings.

Table 3–1 Example of Overlapping Tables

Hotel Reservation Operations	Front Desk Operations
Table 1	Table 12
Table 7	Table 14
Table 15	**Table 15**
Table 11	Table 16
Table 19	**Table 19**
Table 20	**Table 20**

Your objective is to identify overlaps that can cause global conflicts and thus might adversely affect application performance. In this example, both functions

concurrently access three tables. The remaining tables that are accessed exclusively require fewer resources.

Step 3.2: Define the Access Type for Each Overlap

Determine the access type for each overlap as shown in Table 3–2.

Table 3–2 Example of Table Access Types

Overlap Access Type by Reservations	Overlapping Tables	Overlap Access Type by Front Desk
S (Select)	Table 15	S
I (Insert)	Table 19	I
U (Update)	Table 20	U

In this example, both functions access:

- Table 15 for selects
- Table 19 for inserts
- Table 20 for updates

Step 3.3: Identify Transaction Volumes

Estimate the number of transactions that you expect the overlaps to generate as shown in Table 3–3.

Table 3–3 Example of Table Transaction Volumes

Transaction Overlap by Reservations	Overlaps	Transaction Overlap by Front Desk
S **(10 per second)**	Table 15	S **(50 per second)**
I **(100 per second)**	Table 19	I **(10 per second)**
U **(50 per second)**	Table 20	U **(90 per second)**

Given these transaction volumes, the overlap tables can be a performance problem. However, if the application infrequently accesses the tables, the volumes shown in Table 3–3 may not be a problem.

Step 3.4: Classify Overlaps

Use the following criteria to determine how to improve the scalability of tables accessed by more than one functional area:

- Ignore non-overlapping tables, select-only overlaps, and low-frequency overlaps. From the previous example, you would ignore tables 1, 7, 11, 12, 14, 15, and 16.
- Partition your application to minimize the number of high frequency overlaps occurring from different instances. In other words, try to place functional areas subject to high frequency overlaps in the same instance.
- Remaining overlaps can be resolved using other techniques like departmental partitioning or physical table partitioning.
- Keep in mind that some overlapping occurs in any system, and in most cases, Cache Fusion provides acceptable performance with no need for code changes.
- You can minimize the effect of high frequency `INSERT` overlaps by using free list groups in the insert intensive tables.

See Also : "Using Free List Groups For Concurrent Inserts from Multiple Nodes" on page 4-3

Scaling-Up and Partitioning in Real Application Clusters

If you have properly partitioned your application for Real Application Clusters, then as the size of your database increases you can maintain the same partitioning strategy and simultaneously achieve optimal performance. The partitioning method to use when adding new functionality depends on the types of data the new functions access. If the functions access disjoint data, then your existing partitioning scheme should be adequate. If the new functions access the same data as the existing functions, then you may need to change your partitioning strategy.

If your application attracts more users than you expected, then you may need to add more instances. Adding a new instance can also require that you repartition your application.

Before adding instances to your Real Application Clusters environment, analyze the new instance's data access requirements. If the new instance accesses its own subset of data, or data that is not accessed by existing instances, then your current partitioning strategy should adequately prevent data contention. However, if the new instance accesses existing data, consider the following issues:

- If you are adding new functionality to the new instance and the new functionality requires access to existing tables, then consider revising your partitioning strategy.
- You may also need to alter your partitioning strategy if you reassign some users of an existing application to the additional instance.

> **See Also:** *Oracle9i Real Application Clusters Administration* for information about adding instances

4

Database Design Techniques for Real Application Clusters

This chapter describes database design techniques for **Oracle Real Application Clusters** environments. The sections in this chapter include:

- Principles of Database Design for Real Application Clusters
- Using Free List Groups For Concurrent Inserts from Multiple Nodes
- Using Sequence Numbers in Real Application Clusters
- Tablespace Design in Real Application Clusters
- Index Issues for Real Application Clusters Design
- Object Creation and Performance in Real Application Clusters
- Conclusions and a Summary of Guidelines

Principles of Database Design for Real Application Clusters

When designing database layouts for shared **Oracle Real Application Clusters** databases, remember that accessing globally shared data from multiple **node**s increases transaction processing costs. In other words, multi-node transactions incur more wait time and higher CPU consumption than transactions processed on single node systems. Because of this, if you carefully consider the data access patterns of your applications, your resulting database design will enhance scalability.

In general, you can improve scalability by:

- Assigning transactions with similar data access characteristics to specific nodes
- Creating data objects with parameters that enable more efficient access when globally shared

The most scalable and efficient application designs for clustered systems enable a high degree of transaction affinity to the data that the transactions access. The more local your application's data access, the more efficient your application. In this case, the application minimizes the costs of cross-instance synchronization.

All applications running on multi-node systems have some data with low node affinity. This data is shared across the **cluster** and thus requires synchronization. **Cache Fusion**, however, reduces the costs associated with globally shared database partitions by more efficiently synchronizing this data across multiple nodes.

Some database resources can become critical when certain transactions execute in Real Application Clusters environments. For example, an excessive rate of inter-instance changes to a small number of *hot* data blocks that are in the same table can cause increased inter-instance messaging, context switches, and general processing overhead. If a table has one or more indexes, then the maintenance cost can increase even more due to the relative complexity of index changes.

Searching for free space and allocating it when inserting new data can require access to space management structures, such as segment free lists. Also, you must carefully configure sequence number generation if every node in the cluster uses sequence numbers.

Using Free List Groups For Concurrent Inserts from Multiple Nodes

When data is frequently inserted into a table from multiple nodes and the table is not partitioned, use free list groups to avoid performance issues. In such situations, contention can be due to concurrent access to data blocks, table segment headers, and other global resource demands.

Free list groups separate the data structures associated with the free space management of a table into disjoint sets that are available for individual instances. With free list groups, the contention among processes working on different instances is reduced because data blocks with sufficient free space for inserts are managed separately for each instance.

Another efficient way of avoiding overhead due to concurrency when inserting data from different nodes is to use partitioned tables. However, in this case the application has to make sure that there is affinity between data in the partitions.

Cache Fusion resolves concurrency on shared data between instances by using cache-to-cache transfers. This reduces the overhead associated with maintaining cache coherency. To avoid inter-instance concurrency altogether, use free list groups. However, before building tables, indexes, or clusters with free list groups and free lists, determine whether the feature is useful for the application.

> **See Also:** *Oracle9i Real Application Clusters Concepts* for a conceptual overview of free list groups

Deciding Whether to Create Database Objects with Free List Groups

Before designing your database for a particular application, you should understand how frequently data is added to, modified, or read from your database tables. If you use multiple nodes and users or application modules are routed to a particular node, then concurrency among instances can be low. Thus, you would not have to use any particular design strategy.

Free lists and free list groups are usually needed when random inserts to a table from multiple **instance**s occur frequently. Processes looking for space in data blocks can contend for the same blocks and table headers. Performance can be adversely affected by the degree of concurrency and the overhead of shipping data and header blocks from one instance to another. In these cases, using free list groups can improve performance.

> **See Also:** For more information on partitioning, refer to "Workload Distribution Concepts in Real Application Clusters" on page 3-7.

Identifying Critical Tables Before Migrating to Real Application Clusters

To migrate your application from a single instance environment to Real Application Clusters, identify the tables that are subject to a high rate of inserts. Do this by querying V$SQL and searching for INSERT commands as in the following example:

```
SELECT SUBSTR(SQL_TEXT,80), DECODE(COMMAND_TYPE,2,'INSERT'),EXECUTIONS
FROM V$SQL
WHERE COMMAND_TYPE = 2
ORDER BY EXECUTIONS;
```

Search for the table name in the string for the statements with the highest number of executions. These statements and the indexes that are built on them are candidates for free list groups. Remember to also consider the application partitioning strategy. In other words, a table can be subject to excessive insert rates, but if the INSERT statements always occur from the same instance, you do not need to increase the FREELIST GROUPS parameter for the table. In these cases, changing to free lists would still be beneficial for performance.

Determining FREELIST GROUPS Reorganization Needs

You can monitor free list group performance by examining the rate of cache transfers and **forced disk writes** using the V$CLASS_CACHE_TRANSFER view. V$CLASS_CACHE_TRANSFER view contains information about the number of cache transfers that occurred since instance startup for each class of block. If your output from the following select statement example shows a relatively high amount for segment header and / or free list forced disk writes (more than 5% of the total), then consider changing the FREELIST GROUPS parameter for some tables to improve performance.

```
SELECT CLASS, (X_2_NULL_FORCED_STALE + X_2_S_FORCED_STALE) CACHE_TRANSFER
FROM V$CLASS_CACHE_TRANSFER;
```

Because V$CLASS_CACHE_TRANSFER does not identify cache transfers by object name, you can use other views to identify the objects that significantly contribute to the number of cache transfers. For example, V$CACHE_TRANSFER has information about each block in the buffer cache that is transferred. Block class number 4 identifies segment headers and block class number 6 identifies free list blocks. The

output from the following select statement can show objects that could benefit from increased free list groups values:

```
SELECT NAME, CLASS#, SUM(XNC) CACHE_TRANSFER
FROM V$CACHE_TRANSFER
WHERE CLASS# IN (4,6)
GROUP BY NAME, CLASS#
ORDER BY CACHE_TRANSFER DESC;
```

Note: Certain views such as V$CLASS_CACHE_TRANSFER are only available after you execute the CATCLUST.SQL script.

Creating Tables, Clusters, and Indexes with FREELISTS and FREELIST GROUPS

Create free lists and free list groups by specifying the FREELISTS and FREELIST GROUPS storage parameters in CREATE TABLE, CREATE CLUSTER or CREATE INDEX statements. The database can be opened in either exclusive or shared mode. If you need to use free list groups, then the general rule is to create at least one free list group for each Real Application Clusters instance.

Note: You *cannot* change the value of FREELIST GROUPS with the ALTER TABLE, ALTER CLUSTER, or ALTER INDEX statements unless the table or cluster is exported, dropped, rebuilt, and reloaded. However, you can dynamically change FREELISTS with the ALTER TABLE, ALTER INDEX, or ALTER CLUSTER statements.

FREELISTS Parameter

The FREELISTS parameter specifies the number of free lists in each free list group. The default and minimum value of FREELISTS is 1. The maximum value depends on the data block size. If you specify a value that is too large, then an error message informs you of the maximum value. The optimal value for FREELISTS depends on the expected number of concurrent inserts for each free list group for a particular table.

FREELIST GROUPS Parameter

Each free list group is associated with one or more instances at startup. The default value of FREELIST GROUPS is 1. This means that all existing free lists of a segment

are available to all instances. As mentioned, you would typically set FREELIST GROUPS equal to the number of instances in Real Application Clusters.

Free list group blocks with enough free space for inserts and updates are effectively disjoint once Oracle allocates them to a particular free list group. However, once data blocks that are allocated to one instance are freed by another instance, they are no longer available to the original instance. This might render some space unusable and possibly create a skew.

Note: With multiple free list groups, the free list structure is detached from the segment header and located in the free list block, which is a separate block. This reduces contention for the segment header and provides separate free block lists for instances.

Example The following statement creates a table named department that has seven free list groups, each of which contains four free lists:

```
CREATE TABLE department
        (deptno   NUMBER(2),
         dname    VARCHAR2(14),
         loc      VARCHAR2(13) )
        STORAGE ( INITIAL 100K        NEXT 50K
                  MAXEXTENTS 10       PCTINCREASE 5
                  FREELIST GROUPS 7   FREELISTS 4 );
```

Creating FREELISTS and FREELIST GROUPS for Clustered Tables

Use clustered tables to store records from different tables if the records are frequently accessed as a group by one or more SELECT statements. Using clustered tables can thus improve performance by reducing the overhead for processing reads. However, clustered tables may be less useful for DML statements.

You cannot specify FREELISTS and FREELIST GROUPS storage parameters in the CREATE TABLE statement for a clustered table. Instead, specify free list parameters for the *entire* cluster rather than for individual tables. This is because clustered tables use the storage parameters of the CREATE CLUSTER statement.

Real Application Clusters allows clusters (other than hash clusters) to use multiple free lists and free list groups. Some hash clusters can also use multiple free lists and free list groups if you created them with a user-defined key for the hashing function and the key is partitioned by instance.

Note: Using the TRUNCATE TABLE *table_name* REUSE STORAGE syntax removes extent mappings for free list groups and resets the **high water mark** to the beginning of the first extent.

Creating FREELISTS for Indexes

You can also use the FREELISTS and FREELIST GROUPS parameters in the CREATE INDEX statement. However, you should be aware that inserting into an index differs from inserting into a table because the block Oracle uses is determined by the index key value.

For example, assume you have a table with multiple free list groups that also has an index with multiple free list groups. If two sessions connect to different instances and insert rows into that table, then Oracle uses different blocks to store the table data. This minimizes cache block transfers for the affected data segment. However, index segment cache block transfers can still occur if these sessions insert similar index key values. Therefore, you can only anticipate a slight reduction in cache transfers for the index segment header because Oracle must use more header blocks to store the index free lists.

See Also: *Oracle9i SQL Reference* for more information on the SQL mentioned in this section

Associating Instances and User Sessions with Free List Groups

When Oracle creates an object with multiple free list groups, the number of a free list group block becomes part of the object's data dictionary definition. It is important to realize that instances and users need to be associated with a free list group block. You can establish this association statically by assigning a fixed **instance number** to an instance using an initialization parameter, or by specifying the instance number in DDL statements.

The following topics describe:

- Associating Instances with Free List Groups
- Associating User Processes with Free List Groups

Associating Instances with Free List Groups

You can associate an instance with free list groups as follows:

INSTANCE_NUMBER parameter	You can use various SQL clauses with the INSTANCE_NUMBER initialization parameter to associate extents of data blocks with instances.
SET INSTANCE clause	You can use the SET INSTANCE clause of the ALTER SESSION statement to ensure a session uses the free list group associated with a particular instance regardless of the instance to which the session is connected. For example: `ALTER SESSION SET INSTANCE = inst_no`

The SET INSTANCE clause is useful when an instance fails and users re-connect to other instances. For example, consider a database where space is preallocated to the free list groups in a table. If an instance fails and all the users are failed over to other instances, then their session can be set to use the free list group associated with the failed instance.

If you omit the SET INSTANCE clause, then the failed over sessions would start inserting data into blocks and extents allocated to the instance they failed over to. Later, when the failed instance is restored and the users connect to it again, the data they inserted would be part of a set of blocks associated with the other instance's free list group. Thus, inter-instance communication could increase.

Associating User Processes with Free List Groups

User processes are automatically associated with free lists based on the Oracle process ID of the process in which they are running as shown in the following example:

$$(oracle\_pid \bmod \#free\_lists\_for\_object) + 1$$

You can use the ALTER SESSION SET INSTANCE statement to use the free list group associated with a particular instance.

Preallocating Extents

Before Oracle inserts rows into a table, the table only has an initial extent with a number of free blocks allocated to it. Otherwise the table is empty. Therefore, you should attempt to preallocate space for the table in a free list group. This guarantees an optimal allocation of extents containing free blocks to the free list groups, and therefore to the instances. Preallocation also avoids extent allocation overhead.

The advantage of doing this is that the physical storage layout can be determined in advance. Moreover, the technique of allocating extents enables you to select the physical file or volume from which the new extents are allocated. However, you should consider whether and how to implement the ALLOCATE EXTENT clause and a few Oracle initialization parameters when you preallocate as described in the following paragraphs:

- Preallocating Extents with The ALLOCATE EXTENT Clause
- Preallocating Extents by Setting MAXEXTENTS, MINEXTENTS, and INITIAL Parameters
- Preallocating Extents by Setting the INSTANCE_NUMBER Parameter
- Extent Preallocation Examples

Preallocating Extents with The ALLOCATE EXTENT Clause

The ALLOCATE EXTENT clause of the ALTER TABLE or ALTER CLUSTER statement enables you to preallocate an extent to a table, index, or cluster with parameters to specify the extent size, **datafile**, and a group of free lists with which to associate the object.

Exclusive and Shared Modes and the ALLOCATE EXTENT Clause You can use the ALTER TABLE (or CLUSTER) ALLOCATE EXTENT statement while the database is running in exclusive mode, as well as in shared mode. When an instance runs in exclusive mode, the instance still follows the same rules for locating space. A transaction can use the master free list or the specific free list group for that instance.

The SIZE Parameter and the ALLOCATE EXTENT CLAUSE The SIZE parameter of the ALLOCATE EXTENT clause is the extent size in bytes, rounded up to a multiple of the block size. If you do not specify SIZE, then Oracle calculates the extent size according to the values of the NEXT and PCTINCREASE storage parameters.

Oracle does not use the value of SIZE as a basis for calculating subsequent extent allocations, which are determined by the values set for the NEXT and PCTINCREASE parameters.

The DATAFILE Parameter and the ALLOCATE EXTENT Clause This parameter specifies the datafile from which to take space for an extent. If you omit this parameter, then Oracle allocates space from any accessible datafile in the **tablespace** containing the table.

The filename must exactly match the string stored in the **control file**; it is case-sensitive. You can check the FILE_NAME column of the DBA_DATA_FILES data dictionary view for this string.

The INSTANCE Parameter and the ALLOCATE EXTENT Clause This parameter assigns the new space to the free list group associated with the **instance number** integer. At startup, each instance acquires a unique instance number that maps the instance to a group of free lists. The lowest instance number is 1, not 0; the maximum value is operating system-specific. The syntax is:

```
ALTER TABLE tablename ALLOCATE EXTENT (... INSTANCE n )
```

where *n* maps to the free list group with the same number. If the instance number is greater than the number of free list groups, then it is hashed as follows to determine the free list group to which it is assigned:

$$modulo(n,\#\_freelistgroups) + 1$$

If you do not specify the INSTANCE parameter, then the new space is assigned to the table but not allocated to any group of free lists. Such space is included in the master free list of free blocks as needed when no other space is available.

Note: Use a value for INSTANCE that corresponds to the number of the free list group you wish to use—rather than the actual instance number.

See Also: *Oracle9i Real Application Clusters Administration* for more information about the INSTANCE parameter

Preallocating Extents by Setting MAXEXTENTS, MINEXTENTS, and INITIAL Parameters

You can prevent automatic extent allocations by preallocating extents to free list groups associated with particular instances, and by setting MAXEXTENTS to the current number of extents (preallocated extents plus MINEXTENTS). You can minimize the initial allocation when you create the table or cluster by setting

MINEXTENTS to 1 (the default) and by setting INITIAL to its minimum value (two data blocks, or 10K for a block size of 2048 bytes).

To minimize contention among instances for data blocks, create multiple datafiles for each table and associate each instance with a different file.

If you expect to increase the number of nodes in your system, then allow for additional instances by creating tables or clusters with more free list groups than the current number of instances. You do not have to allocate space to those free list groups until it is needed. Only the master free list of free blocks has space allocated to it automatically.

To associate a data block with a free list group, either bring the data block below PCTUSED by a process running on an instance using that free list group, or specifically allocate the block to that free list group. Therefore, a free list group that is never used does not leave unused free data blocks.

Preallocating Extents by Setting the INSTANCE_NUMBER Parameter

The INSTANCE_NUMBER initialization parameter enables you to start an instance and ensure that it uses the extents allocated to it for inserts and updates. This ensures that it does not use space allocated for other instances. The instance cannot use data blocks in another free list belonging to another instance, unless the instance is restarted with the other instance's INSTANCE_NUMBER. However, you can override the instance number during a session by using an ALTER SESSION statement.

Extent Preallocation Examples

This section provides examples in which extents are preallocated.

Example 1 The following example statement allocates an extent for table DEPARTMENT from the datafile DEPT_FILE7 to instance number 7:

```
ALTER TABLE department
ALLOCATE EXTENT ( SIZE 20K
                  DATAFILE 'dept_file7'
                 INSTANCE 7);
```

Example 2 The following SQL statement creates a table with three free list groups, each containing ten free lists:

```
CREATE TABLE table1 ... STORAGE (FREELIST GROUPS 3 FREELISTS 10);
```

The next SQL statement then allocates new space, dividing the allocated blocks among the free lists in the second free list group:

```
ALTER TABLE table1 ALLOCATE EXTENT (SIZE 50K INSTANCE 2);
```

In a Real Application Clusters environment that runs more instances than the value you have set for the FREELIST GROUPS storage parameter, multiple instances share the new space allocation. In this example, every third instance to start up is associated with the same group of free lists.

Example 3 The following CREATE TABLE statement creates a table named EMPLOYEE with one initial extent and three groups of free lists. The three ALTER TABLE statements allocate one new extent to each group of free lists:

```
CREATE TABLE employee ...
 STORAGE ( INITIAL 4096
           MINEXTENTS 1
           MAXEXTENTS 4
           FREELIST GROUPS 3 );
ALTER TABLE employee
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile1' INSTANCE 1 )
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile2' INSTANCE 2 )
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile3' INSTANCE 3 );
```

To prevent automatic allocations, MAXEXTENTS is set to 4 which is the sum of the values of MINEXTENTS and FREELIST GROUPS.

When you need additional space beyond this allocation, use the ALTER TABLE statement to increase MAXEXTENTS before allocating additional extents. For example, if the second group of free lists requires additional free space for inserts and updates, you could set MAXEXTENTS to 5 and allocate another extent for that free list group:

```
ALTER TABLE employee ...
 STORAGE ( MAXEXTENTS 5 )
 ALLOCATE EXTENT ( SIZE 100K DATAFILE 'empfile2' INSTANCE 2 );
```

Using Sequence Numbers in Real Application Clusters

When designing applications for Real Application Clusters, use Oracle sequence numbers whenever possible. To optimize sequence number use, each instance's cache must be large enough to accommodate the sequences. The default cache size holds 20 sequence numbers. To increase this, for example to hold 200, use this syntax:

```
ALTER SEQUENCE sequence_name CACHE 200;
```

Using the ordering feature suppresses caching in Real Application Clusters. But note that it is normal to lose some numbers after executing the SHUTDOWN command or after instance failures. This is true even in single instance configurations.

If ordering is essential, then you may need to disable sequence caching. In this case, expect some performance overhead.

> **See Also:** *Oracle9i Database Concepts* for more information about sequences

Detecting Global Conflicts for Sequences

If sequences are insufficiently cached or not cached at all, then performance problems can result with an increase in service times. If you experience performance problems, then examine the statistics in the V$SYSTEM_EVENT view as described in the following two points to determine whether the problem is due to the use of Oracle sequences:

- A problem with sequences appears in V$SYSTEM_EVENT as extended average wait times for *row cache locks* in the range of a few hundred milliseconds. The proportion of time waited for row cache locks to the total time waited for non-idle events will be relatively high.
- For the DC_SEQUENCES parameter, the ratio of DLM_CONFLICTS to DLM_REQUESTS will be high. If this ratio exceeds 10 to 15%, and the row cache lock wait time is a significant portion of the total wait time, then it is likely that the service time deterioration is due to insufficiently cached sequences.

Using Database Tables to Generate Sequence Numbers

If your application cannot afford to lose a sequence number, then you may want to implement sequences by storing them in database tables. However, there is significant performance overhead associated with the mechanism required for

implementing this strategy. This is true even in single instance environments. As a general recommendation, rows storing sequence numbers should be locked for only a very brief period.

In Real Application Clusters, there can be additional overhead associated with the cache coherence needed for buffers storing sequence numbers. If a single data block stores several sequence numbers, and if more than one instance needs those sequence numbers, then the data block can be frequently transferred among the instances.

To minimize that overhead, set PCTFREE to a very high value so Oracle stores only a single row of the table containing the sequence numbers in each data block. In that case, the cache transfers only occur when the instances concurrently request the same sequence number.

Tablespace Design in Real Application Clusters

Your goal in tablespace design is to group database objects according to their data access distribution patterns. If you consider the dependency analyses and transaction profiles of your database objects, then you can divide tablespaces into containers for the following objects:

- Frequently and randomly modified tables and indexes belonging to particular functional areas
- Frequently and randomly modified tables and indexes with a lower probability of having affinity to any functional area
- Tables and indexes that are mostly READ or READ-ONLY and infrequently modified

Consider the following additional criteria for separating database objects into tablespaces:

- Tables should be separated from indexes
- Assign read-only tables to READ-ONLY tablespaces
- Group smaller reference tables in the same tablespace

Grouping database objects that belong to different functional areas into different tablespaces using this strategy can improve dynamic resource mastering. This works best if you adopt a functional partitioning strategy as described in Chapter 3. Oracle's dynamic resource re-mastering by datafiles algorithm re-distributes GCS resources where they are needed most. This re-mastering strategy improves resource operations efficiency. That is, Oracle re-masters resources to the instance

with which the resources are most closely associated based on access patterns. As a result, resource operations after re-mastering require minimal communication with remote instances through the **Global Enqueue Service (GES)** and **Global Cache Service (GCS)**.

In rare cases, you can further reduce GCS traffic by changing the default resource control policy for some tablespaces, as described in Appendix A, "Configuring Multi-Block Lock Assignments (Optional)".

> **See Also:** *Oracle9i Real Application Clusters Concepts* for more information about dynamic resource remastering

Extent Management and Locally Managed Tablespaces

Allocating and deallocating extents are expensive operations that you should minimize. Most of these operations in Real Application Clusters require inter-instance coordination. In addition, a high rate of extent management operations can more adversely affect performance in Real Application Clusters environments than in single instance environments. This is especially true for dictionary managed tablespaces.

Identifying Extent Management Issues

If the "row cache lock" event is a significant contributor to the non-idle wait time in `V$SYSTEM_EVENT`, then there is contention in the data dictionary cache. Extent allocation and deallocation operations could cause this.

`V$ROWCACHE` provides data dictionary cache information for `DC_USED_EXTENTS` and `DC_FREE_EXTENTS`. This is particularly true when the values for `DLM_CONFLICTS` for those parameters increase significantly over time. This means that excessive extent management activity is occurring.

Minimizing Extent Management Operations

Proper storage parameter configuration for tables, indexes, temporary segments, and **rollback segments** decreases extent allocation and deallocation frequency. Do this using the `INITIAL`, `NEXT`, `PCTINCREASE`, `MINEXTENTS`, and `OPTIMAL` parameters.

Using Locally Managed Tablespaces

You can greatly reduce extent allocation and deallocation overhead if you use locally managed tablespaces. For optimal performance and space use, segments in

locally managed tablespaces should ideally have similar space allocation characteristics. This enables you to create the tablespace with the proper uniform extent size that corresponds to the ideal extent size increment calculated for the segments.

For example, you could put tables with relatively high insert rates in a tablespace with a 10MB uniform extent size. On the other hand, you can place small tables with limited DML activity in a tablespace with a 100K uniform extent size. For an existing system where tablespaces are not organized by segment size, this type of configuration can require significant reorganization efforts with limited benefits. For that reason, the compromise is to create most of your tablespaces as locally managed with `AUTOALLOCATE` instead of `UNIFORM` extent allocation.

> **See Also:** *Oracle9i SQL Reference* for more information about the `AUTOALLOCATE` and `UNIFORM` clauses of the `CREATE TABLESPACE` statement

Index Issues for Real Application Clusters Design

In high volume OLTP systems, inter-instance concurrent index block accesses can increase the cost of Real Application Clusters processing. This is because the commonly used B+-Tree index structures usually contribute to higher Cache Fusion activity. A *right-growing* tree can incur frequent cache transfers of one particular leaf block.

While traversing the tree structure, branch blocks might have to be requested from another instance that recently modified them. Leaf block splits are vulnerable because three blocks need to be modified in one transaction. For very high transaction volumes occurring from different instances, you may need to reduce inter-instance concurrent changes to:

- Leaf and branch blocks
- Root blocks
- Index segment headers

The following section addresses how to reduce leaf, branch, and root block contention. You can reduce index segment header concurrent changes by using free list groups as described under the heading "Using Free List Groups For Concurrent Inserts from Multiple Nodes" on page 4-3.

Reducing Inter-Instance Concurrent Changes To Index Blocks

This section describes the following four strategies to isolate or distribute access to different parts of an index and to improve performance:

- Using Reverse Key Indexes to Distribute Index Access
- Assigning Different Subsequences to Each Instance to Reduce Index Contention
- Using INSTANCE_NUMBER to Generate Index Keys
- Reducing Index Contention by Partitioning Tables by Range

Using Reverse Key Indexes to Distribute Index Access

Use **reverse key indexes** to avoid right-growing index trees. By reversing the keys, you can achieve a broader spread of index keys over the leaf blocks of an index and thus reduce the probability of accessing the same leaf and branch blocks from multiple instances.

> **Note:** Reverse key indexes do not allow index range scans so carefully consider this before using them.

Assigning Different Subsequences to Each Instance to Reduce Index Contention

For indexes based on sequence numbers, you can assign different subsequences to each instance. In the case of database objects that can be partitioned based on certain characteristics, this might adequately distribute the access patterns.

Using INSTANCE_NUMBER to Generate Index Keys

For other sequentially assigned values, adjust the index value and use `INSTANCE_NUMBER` to generate the index key, as shown in the following formula:

$$index\ key = (instance\_number - 1) * 100000 + Sequence\ number$$

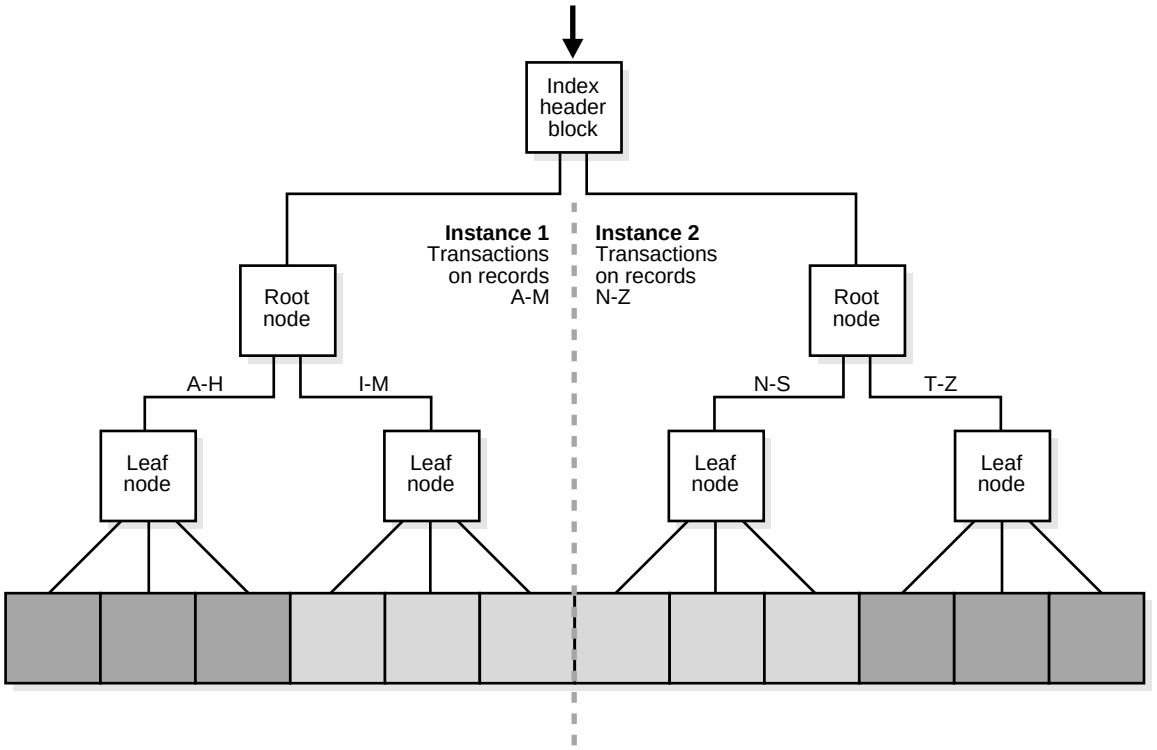
Reducing Index Contention by Partitioning Tables by Range

Another effective way to reduce index contention is to partition tables by range and to create local indexes on them.

See Also: "Scaling Applications for Real Application Clusters" on page 3-1 for guidelines on physical table partitioning implementation.

Figure 4–1 shows how transactions operating on records stored in tables partitioned by range can minimize leaf and branch block contention.

Figure 4–1 Node Affinity for Transactions Against Tables Partitioned by Range



Minimizing Table Locks to Optimize Performance

In Real Application Clusters, Oracle uses inter-instance communication to globally coordinate table locks. Because most applications do not need to lock entire tables, you can disable table locks to improve locking efficiency with minimal adverse side-effects. There are two methods for disabling table locks as described under the following headings:

- Disabling Table Locks for Individual Tables
- Setting DML_LOCKS to Zero

> **Note:** Oracle cannot execute DDL statements against tables with disabled locks.

Disabling Table Locks for Individual Tables

To prevent users from acquiring table locks, use the following statement:

```
ALTER TABLE table_name DISABLE TABLE LOCK
```

Users attempting to lock tables with disabled locks receive an error. To re-enable table locking, use the following statement:

```
ALTER TABLE table_name ENABLE TABLE LOCK
```

This syntax forces all currently executing transactions to commit before enabling the table lock. The statement does not wait for new transactions to start after issuing the ENABLE statement.

To determine whether a table has its table lock enabled or disabled, query the TABLE_LOCK column in the USER_TABLES data dictionary table. If you have select privilege on DBA_TABLES or ALL_TABLES, then query the table lock state of other user's tables.

Setting DML_LOCKS to Zero

You can set table locks set for an entire instance using the DML_LOCKS initialization parameter. If you do not need to use the DROP TABLE, CREATE INDEX, and LOCK TABLE statements, then set DML_LOCKS to zero to minimize lock conversions and achieve maximum performance.

> **Note:** If you set DML_LOCKS to zero on one instance, then you must set it to zero on *all* instances. If you use non-zero values with the DML_LOCKS parameter, the values need *not* be identical on all instances.

SQL*Loader checks the flag to ensure that there is not a non-parallel direct load running against the same table. This forces Oracle to create new extents for each session.

> **See Also:** *Oracle9i Database Utilities* for more information on SQL*Loader

Object Creation and Performance in Real Application Clusters

As a general database design rule, you should only use DDL statements for maintenance tasks, not during normal system operations. Therefore, in most systems, the frequency of new object creation and other DDL statements should be very small.

However, if your application frequently creates objects, some performance degradation may occur. This is because object creation requires inter-instance coordination. A high ratio of DLM_CONFLICTS to DLM_REQUESTS on the DC_OBJECT_IDS row cache in V$ROWCACHE, along with excessive wait times for the row cache lock event in V$SYSTEM_EVENT, indicates that different instances in your cluster are issuing significant amounts of concurrent DDL statements.

To improve object creation performance is such situations, set event 10297 so that it caches OBJECT_ID values. This improves concurrent object creation. To set event 10297, add the following line to your **initialization parameter file**:

```
event="10297 trace name context forever, level 1"
```

If you set the additional level argument to 1, then the caching behavior is automatically adjustable. Otherwise, you can set level to the desired cache size.

Conclusions and a Summary of Guidelines

Cache Fusion introduces an improved diskless algorithm that handles cache coherency more efficiently than Oracle's earlier architectures. This enables you to implement simpler database designs while achieving optimal performance.

Response time and throughput requirements ultimately determine whether you should implement a partitioning strategy and how stringent your strategy needs to be. Response time and throughput requirements also determine how much effort you should invest to achieve an optimal database design.

A careful analysis of your system's workload should serve as the optimal basis for allocating work to particular instances. This analysis should consider:

- System resource consumption
- Data access distributions by functional area
- Functional dependencies

Moreover, implementing a strategy that considers these points makes your system more robust and thus more scalable.

Generally speaking, 80% or more of your overhead results from 20% or less of a given workload. If you first attempt to deal with the 20% by observing some simple guidelines, then you can produce tangible benefits with minimal effort. You can address these workload problems by implementing any or all of the following:

- Define free list groups for partitioned as well as non-partitioned data that is frequently modified.
- Use read-only tablespaces wherever data remains constant.
- Use locally managed tablespaces to reduce extent management costs.
- Use Oracle sequences to generate unique numbers and set the CACHE parameter to a high value if needed.
- If possible, reduce concurrent changes to index blocks. However, if index key values are not modified by multiple instances, or if the modification rate is not excessive, the overhead may be acceptable. In extreme cases, you can apply techniques like physical table partitioning.

Part III

Real Application Clusters Performance Monitoring and Tuning

Part Three describes how to monitor performance statistics and adjust parameters to improve Real Application Clusters performance. Part Three contains the following chapters:

- Chapter 5, "General Tuning Recommendations for Real Application Clusters"
- Chapter 6, "Tuning Real Application Clusters and Inter-Instance Performance"

> **See Also:** Chapter 7, "Monitoring Performance with Oracle Performance Manager" for more information on using Oracle Enterprise Manager to monitor and tune Real Application Clusters databases

5

General Tuning Recommendations for Real Application Clusters

This chapter provides an overview of tuning issues for **Oracle Real Application Clusters**. It presents a general methodology for tuning applications for Real Application Clusters environments and includes the following topics:

- Overview of Tuning Real Application Clusters
- Statistics for Monitoring Real Application Clusters Performance
- Using Views to Evaluate Real Application Clusters Performance
- Measuring Workload Performance in Real Application Clusters
- General Tuning Recommendations for Workload Performance
- Determining the Costs of Synchronization In Real Application Clusters
- Measuring Global and Local Work Ratios in Real Application Clusters
- Calculating the Global Cache Synchronization Costs Due to Contention in Real Application Clusters
- Resolving Performance Problems in Real Application Clusters-Based Applications
- Advanced Queuing and Real Application Clusters

> **See Also :** Chapter 7, "Monitoring Performance with Oracle Performance Manager" for more information on using **Oracle Enterprise Manager** to monitor and tune Real Application Clusters databases

Overview of Tuning Real Application Clusters

Cache Fusion resolves the types of inter-instance contention that were once responsible for most Real Application Clusters overhead: read / write and write / write contention. Because Cache Fusion virtually eliminates **forced disk writes**, release 1 (9.0.l) of Real Application Clusters requires less tuning than previous releases to achieve scalability.

Many single **instance** tuning practices are useful for Real Application Clusters applications. However, you must also effectively size the buffer cache and shared pool. You must also tune your shared disk subsystems with Real Application Clusters-specific goals in mind. In addition, you should collect Real Application Clusters-specific statistics and monitor views using the methods described in this chapter.

With experience, you can anticipate many performance problems before deploying Real Application Clusters applications. However, even the most effective tuning cannot overcome problems caused by poor analysis or database and application design flaws. Therefore, make sure you have thoroughly considered the issues discussed in Part II of this book, "Scaling Applications and Designing Databases for Real Application Clusters", before evaluating and tuning performance.

Statistics for Monitoring Real Application Clusters Performance

This section provides a brief description of Real Application Clusters application performance statistics. Topics in this section include:

- The Content of Real Application Clusters Statistics
- Recording Statistics for Tuning
- Significant Real Application Clusters Statistics

Oracle maintains most statistics in the local **System Global Area (SGA)**. Access these statistics using SQL statements against V$ views as described in this chapter.

The Content of Real Application Clusters Statistics

Real Application Clusters-specific statistics appear as message request counters or timed statistics. Message request counters include statistics showing the number of a certain type of block mode conversion. Timed statistics, for example, reveal the total or average time waited for read and write I / O on particular operations.

Recording Statistics for Tuning

Oracle Corporation recommends that you record statistics about the rates at which certain events occur. Also record information about specific transactions within your Real Application Clusters environment. Do this with utilities such as Oracle9*i* Statspack or UTLBSTAT and UTLESTAT by recording statistics over a period of time to capture them for specific measurement intervals. In addition, Oracle9*i* Statspack has a page dedicated to displaying Real Application Clusters performance information.

These utilities compute statistics counts per second and per transaction. You can also use them to measure the number of statement executions and the number of business transactions that applications submit.

For example, an insurance application might define a transaction as the processing of an insurance quote. This transaction might be composed of several DML operations and queries. If you know how many of these quotes your system processes in a certain time interval, then divide that value by the number of quotes completed in the interval. Do this over a period of time to gauge performance.

Performance trends in application profiles appear in terms of the resources used per transaction and the application workload. The counts per transaction are useful for detecting changing workload patterns, while rates indicate the workload intensity.

Tracing Execution History with the TRACE_ENABLED Parameter

You can trace the execution history of Oracle using the TRACE_ENABLED parameter. The TRACE_ENABLED parameter is set to true by default to control tracing of the execution history, or code path, of Oracle. Oracle Support uses this information for debugging.

In Real Application Clusters, you must set this parameter to the same value for all instances. You can set TRACE_ENABLED within your **initialization parameter file**, or by using the ALTER SYSTEM SET statement.

When TRACE_ENABLED is set to `true`, Oracle records information in specific files when errors occur. Table 5–1 shows the types of files and the UNIX default destination directories in which Oracle records the execution history.

Table 5–1 Memory Trace Files and Their Locations

Trace File Type	Destination Directory
User	$ORACLE_HOME/rdbms/log/user_dump_dest
Background	$ORACLE_HOME/rdbms/log/background_dump_dest
Core	$ORACLE_HOME/dbs/core_dump_dest

Oracle records this information for all instances, even if only one instance terminates. This allows Oracle to retain diagnostics for the entire **cluster**.

Although the overhead incurred from this processing is not excessive, you can improve performance by setting TRACE_ENABLED to false. You might do this, for example, to meet high-end benchmark requirements. However, if you leave this parameter set to false, you may lose valuable diagnostic information. Therefore, *always* set TRACE_ENABLED to true to trace system problems and to reduce diagnostic efforts in the event of unexplained instance failures.

Significant Real Application Clusters Statistics

The most significant statistics for Real Application Clusters are:

- Cache-related statistics such as **consistent gets**, **db block gets**, and **db block changes**
- Cache Fusion related statistics, such as **global cache current block receive time** or **global cache current block send time**
- global cache lock open and convert requests, and global cache wait times, such as **global cache gets**, **global cache converts**, and waits for events such as Null-to-X conversions
- I/O statistics such as **physical reads**, **physical writes**, **DBWR cross-instance writes**, and wait times for reads and writes

Two of the most important views for displaying Real Application Clusters-specific statistics are V$SYSSTAT and V$SYSTEM_EVENT. The next section provides more details about the contents of these and other views for tuning Real Application Clusters.

Using Views to Evaluate Real Application Clusters Performance

This section describes the content and use of several views for monitoring Real Application Clusters. The topics in this section are:

- Using V$SYSSTAT for Real Application Clusters Statistics
- Using V$SYSTEM_EVENT for Real Application Clusters Statistics
- Using Other Views to Obtain Real Application Clusters Statistics

Using V$SYSSTAT for Real Application Clusters Statistics

The V$SYSSTAT view provides statistics about your entire Real Application Clusters environment. That is, the statistics are global. Real Application Clusters-specific statistics in V$SYSSTAT belong to classes 8, 32, and 40 as shown in Table 5–2.

Table 5–2 Statistics and their Classes in V$SYSSTAT

Statistic Name	Class
consistent gets	8
db block gets	8
db block changes	8
gcs messages sent	32
ges messages sent	32
physical reads	8
physical writes	8
DBWR cross-instance writes	40
global lock sync gets	32
global lock async gets	32
global lock get time	32
global lock sync converts	32
global lock async converts	32
global lock convert time	32
global lock releases	32
global cache gets	40

Table 5–2 Statistics and their Classes in V$SYSSTAT

Statistic Name	Class
global cache get time	40
global cache converts	40
global cache convert time	40
global cache cr blocks received	40
global cache cr block receive time	40
global cache current blocks received	40
global cache current block receive time	40
global cache cr blocks served	40
global cache cr block build time	40
global cache cr block flush time	40
global cache cr block send time	40
global cache current blocks served	40
global cache current block pin time	40
global cache current block flush time	40
global cache current block send time	40
global cache freelist waits	40
global cache defers	40
global cache convert timeouts	40
global cache blocks lost	40
global cache blocks corrupt	40
global cache prepare failures	40

Using V$SYSTEM_EVENT for Real Application Clusters Statistics

The V$SYSTEM_EVENT view provides statistics about the frequency with which Oracle processes have to wait for events. They also show the number of timeouts for these events and their cumulative and average durations. These events are also referred to as *waits*. The statistics in V$SYSTEM_EVENT that are relevant to Real Application Clusters are:

- buffer busy
- buffer busy due to global cache
- cr request retry
- db file parallel write
- db file scattered read
- db file sequential read
- enqueue
- global cache cr request
- global cache busy
- global cache null to s
- global cache null to x
- global cache open s
- global cache open x
- global cache s to x
- KJC: Wait for msg sends to complete
- library cache pin
- log file sync
- row cache lock

You can also analyze operating system statistics that reveal CPU use and disk I/O. The procedures in this chapter can help you analyze these statistics to determine the amount of CPU that Oracle uses for certain background processes such as:

- **Global Cache Service Processes (LMSn)**
- **Database Writer (DBWn)**
- **Global Enqueue Service Daemon (LMD)**

Using Other Views to Obtain Real Application Clusters Statistics

Other important statistics appear in the following views:

- V$CACHE
- V$LOCK_ACTIVITY
- V$GES_STATISTICS

The statistics in these views that are relevant to Real Application Clusters relate to the following performance issues:

- Buffer cache use
- Types of block mode conversions
- Resource control activity with regard to block classes and files
- Number of blocked convert requests
- Messages the **Global Cache Service (GCS)** sends and receives

---

Note: The FORCED WRITES columns in V$CACHE should always be 0 (zero) in Oracle9*i*.

Measuring Workload Performance in Real Application Clusters

In Real Application Clusters, application performance and scalability are determined by the rate and cost of synchronization among **instances**. You can measure the costs by identifying how effectively transactions use CPU resources to:

- Send and receive messages
- Maintain block modes and resources required to guarantee global cache coherency

You can also calculate CPU time spent:

- At the **Inter-Process Communication (IPC)** layer
- To process block requests
- In processing read and write I/O caused by synchronization between **nodes**
- For application processing, such as parsing SQL statements, fetching rows, and sorting

Statistics about these events appear in V$ tables such as V$SYSTEM_EVENT. The response time for each transaction depends on the number of resource operations. Response time also depends on the time required to process the instructions, plus the delay or wait time for each request. Disk I/O incurred when using non-default resource control increases response time.

Contention on certain resources adds to the cost of each measurable component. For example, the following events can result in waits for busy buffers:

- Frequent I/O requests to certain disks
- Contention for the same data or index blocks by local and remote transactions

This can in turn increase costs for each transaction and increase your operating system overhead.

As described in Chapter 3, you can create Real Application Clusters applications that are more scalable and that perform well by designing them to minimize inter-node synchronization and communication requirements. Scalable applications meet user service level requirements by minimizing either the rate or the cost of synchronization.

General Tuning Recommendations for Workload Performance

Consider some of the tuning recommendations to improve workload performance as described in this section. The topics in this section are:

- Measuring Workload Performance
- Using V$CLASS_CACHE_TRANSFER and V$FILE_CACHE_TRANSFER for Real Application Clusters Statistics
- Identifying Contended Objects with V$CACHE, V$CACHE_TRANSFER, and V$BH
- Estimating I/O Synchronization Costs

Measuring Workload Performance

In Real Application Clusters, application performance and scalability are determined by the rate and cost of synchronization among instances. Overhead from the Global Cache and Global Enqueue Services might occur when transactions wait for I/O events and block access mode convert requests. Conflicts for blocks between local and remote transactions while opening or converting block access modes can increase synchronization costs. For example, you can estimate the cost of a transaction or request using the following formula:

$$CPUapps + CPUsyncio + CPUipc + CPUgcs + CPUges + WAITsyncio + WAITipc + WAITgcs + WAITges$$

Using V$CLASS_CACHE_TRANSFER and V$FILE_CACHE_TRANSFER for Real Application Clusters Statistics

Use the V$CLASS_CACHE_TRANSFER and V$FILE_CACHE_TRANSFER views to determine the rate of cache block transfers and their block access mode conversion rates. V$CLASS_CACHE_TRANSFER provides a summary of forced disk write activity for each block class. V$FILE_CACHE_TRANSFER shows the amount of cache transfer activity occurring in your environment on a per-file basis.

Both of these views have several different types of *forced write* columns. There is one forced write column for each type of conversion. However, the columns in these views that refer to forced writes remain 0 (zero). A non-zero value indicates that cache transfers are occurring.

Identifying Contended Objects with V$CACHE, V$CACHE_TRANSFER, V$BH, and V$FILE_CACHE_TRANSFER

The V$CACHE, V$CACHE_TRANSFER, and V$BH views show the block classes and blocks that Oracle transfers over the **interconnect** on a per-object basis. Use the FORCED_READS and FORCED_WRITES columns in these views to determine which objects and blocks your Real Application Clusters instances use. This information can help characterize your system's workload. The FORCED_WRITES column provides a count of how often a certain block experiences a forced disk write out of a local buffer cache because the current version was requested by another instance.

Also use V$FILE_CACHE_TRANSFER to identify files that experience cache transfers. If a file shows significant cache transfer activity, then it could also mean the file is experiencing excessive forced disk write activity.

Estimating I/O Synchronization Costs

I/O synchronization costs can adversely affect performance. To evaluate the effects of I/O synchronization, use the V$SYSSTAT view for the counts of the following request statistics:

- DBWR cross-instance writes
- Physical writes
- Physical reads

Also refer to the V$SYSTEM_EVENT view for time waited and average waits for the following statistics:

- db file parallel write
- db file sequential read
- db file scattered read

To estimate the time waited for reads incurred by re-reading data blocks that Oracle had to write to disk due to requests from other instances, divide the statistic, for example, the time waited for db file sequential reads, by the percentage of read I/O caused by previous cache flushes as shown in this formula where **lock buffers for**

read is the value for block access mode conversions from N to S derived from V$LOCK_ACTIVITY and *physical reads* is a value from the V$SYSSTAT view:

$$\frac{\textit{lock buffers for read}}{\textit{physical reads}}$$

Similarly, estimate the proportion of the time waited for database file cross-instance writes caused by cache transfers by dividing the db file parallel write time in V$SYSTEM_EVENTS where *DBWR cross-instance writes* and *physical writes* are values from V$SYSSTAT:

$$\frac{\textit{DBWR cross-instance writes}}{\textit{physical writes}}$$

Determining the Costs of Synchronization In Real Application Clusters

This section explains how to determine the cost incurred by synchronization and coherency processing between instances due to additional CPU time, I/O, and global resource processing and contention. To do this, examine Oracle statistics as described in the following sections:

- Calculating CPU Service Time Required
- Measuring Global Cache Coherency and Contention
- Measuring Global and Local Work Ratios in Real Application Clusters
- Calculating the Global Cache Synchronization Costs Due to Contention in Real Application Clusters
- Measuring Global Cache Coherency and Contention

Calculating CPU Service Time Required

To derive the CPU service time required per transaction, divide the CPU used by a session as shown in V$SYSSTAT by the number of user commits or the number of business transactions. Note that this is the amount of time required by the user process to execute in either user or kernel mode. This does not include the time spent by the operating system kernel on behalf of a transaction.

This measure is useful for comparing how single instance environment applications behave when running in exclusive mode as compared to how application run in Real Application Clusters environments. This measure is also useful for comparing the effects of different workloads and application design changes.

Measuring Global Cache Coherency and Contention

Table 5–3 describes some of the statistics in the V$SYSSTAT and V$SYSTEM_EVENT global cache coherency-related views.

Table 5–3 Global Cache Coherency and Contention Views and Their Statistics

View	Statistics
Refer to V$SYSSTAT to count requests for the actions shown to the right.	**global cache gets** (count of new resources opened) **global cache converts** (count of conversions for blocks) **global cache cr blocks received** (count of consistent read buffers received from the Global Cache Service process (LMS)) **global cache cr blocks served** (count of consistent read buffers sent by LMS) **global cache current blocks received** (count of current blocks received) **global cache current blocks served** (count of current buffers sent to remote instance) Note: Also refer to the convert type-specific rows in V$LOCK_ACTIVITY.
Refer to V$SYSSTAT for the amount of time needed for the actions shown to the right.	**global cache get time** **global cache convert time** **global cache cr block receive time** **global cache cr block build time** **global cache cr block flush time** **global cache cr block send time** **global cache current block receive time** **global cache current block pin time** **global cache current block flush time** **global cache current block send time**

View	Statistics
Refer to V$SYSTEM_EVENT for time waited for the events shown to the right.	**cr request retry** **global cache null to X** **global cache null to S** **global cache S to X** **global cache open X** **global cache open S** **global cache cr request** **global cache freelist wait** **global cache bg acks** **global cache pending ast** **global cache retry prepare** **global cache cr cancel wait** **global cache pred cancel wait**

Refer to the following statistics and views for indicators of high contention or excessive delays:

- **global cache cr timeouts** and **global cache convert timeouts** as found in V$SYSSTAT
- **global cache busy** and **buffer busy due to global cache** as found in V$SYSTEM_EVENT

Maintaining Application Profiles per Transaction per Unit of Time

As mentioned, it is useful to maintain application profiles per transaction and per unit of time. This enables you to compare two distinct workloads and to detect workload changes. These rates are also helpful in determining capacities and for identifying throughput issues. To do this, Oracle Corporation recommends that you incorporate the following ratios of statistics in your performance monitoring scripts:

The Global Cache Service (GCS) performs block mode conversions at a per transaction rate of:

- **global cache gets**
- **global cache converts**

Block transfer throughput consists of the per transaction rates of:

- **global cache cr blocks received**
- **global cache cr blocks served**
- **global cache current blocks received**
- **global cache current blocks served**

Or per transaction rates of **cache convert waits** for block mode conversions such as:

- **global cache null to X**
- **global cache null to S**
- **global cache S to X**

Or per transaction rates of **cache open waits** for block mode conversions such as:

- **global cache open X**
- **global cache open S**

Calculate the same statistics per second or per minute by dividing the total counts or times waited by the appropriate measurement interval.

Note: To record timed statistics, set the `TIMED_STATISTICS` parameter to `true`. Oracle records these statistics in hundredths of seconds. If you are not actively collecting statistics, however, set `TIMED_STATISTICS` to `false` to avoid the performance overhead required to collect statistics.

Measuring Global and Local Work Ratios in Real Application Clusters

The percentage of buffers accessed for global work, or the percentage of cache-to-cache block transfers caused by inter-instance synchronization, can be important measures of how efficiently your application processes share data. These percentages can also reveal whether you have designed your database for optimum scalability.

Use the following calculation based on statistics from V$SYSSTAT to determine the percentage of buffer accesses for local operations, in other words, reads and changes of database buffers that are not subject to block mode conversions:

$$\frac{((\text{consistent gets} + \text{db block gets}) - (\text{global cache gets} + \text{global cache converts}) * 100)}{(\text{consistent gets} + \text{db block gets})}$$

Also consider the following formula where **lock buffers for read** is from V$LOCK_ACTIVITY:

$$\frac{(\text{physical reads} - (\text{lock buffers for read})) * 100}{\text{physical reads}}$$

This calculation implies the percent of physical reads by user processes for local work only; it does not refer to forced reads.

In the previous formula, subtract *lock buffers for read* from the physical read statistic in V$SYSSTAT. Base the local write ratio entirely on the corresponding values from V$SYSSTAT.

Apart from determining the proportion of local and global work (the degree of partitioning) you can also use these percentages to detect changing workload patterns. Moreover, these percentages represent the probability that a data block access is either local or global. You can therefore use this information as a rough estimator in scalability calculations.

In addition to these ratios, the proportion of delays due to unavailable resources is easy to derive using the formula:

$$\frac{100 * (\textit{buffer busy due to global cache})}{\textit{buffer busy} + \textit{buffer busy due to global cache}}$$

Or more generally:

$$\frac{100 * (\textit{buffer busy due to global cache})}{\textit{consistent gets} + \textit{db block gets}}$$

When the GCS opens resources and performs block mode conversions, the percentage of waits that are caused by unavailable resources (resources that are being acquired or released) can indicate contention. The contention can be due to either delays in opening or converting block modes or very high contention on small sets of buffers.

Once you identify a problem area, such as a high ratio of global busy waits, converts, and gets, obtain more detail about the problem by referring to the following three views:

- V$FILE_CACHE_TRANSFER
- V$CACHE
- V$CLASS_CACHE_TRANSFER

The statistics in these views identify the files and blocks shared by all instances. These shared files may be responsible for most inter-instance synchronization costs and global cache coherency processing.

You can also query these views to learn about block mode conversions for each file or for each block class. Indirectly, this also indicates the number cache transfers due to the shared cache architecture.

Calculating the Global Cache Synchronization Costs Due to Contention in Real Application Clusters

Reduced throughput and degradation of transaction response times are the result of increased inter-instance synchronization costs. These problems can have several sources as described in this section under the following headings:

- Contention for the Same Data Blocks
- Contention for Segment Headers and Free List Blocks
- Contention for Resources other than Database Blocks
- Contention Problems Specific to Applications Running on Real Application Clusters

Contention for the Same Data Blocks

Contention for the same data blocks occurs if rows commonly accessed from multiple instances are spread over a limited range of blocks. The probability of this happening depends on the access distribution of the data within the table as a result of the application's behavior.

The probability of contention can also depend on the block size. For example, more rows fit into an 8K block than into a 4K block. The PCTFREE that you defined for the table can also affect the level of contention. In fact, database block size and PCTFREE can be part of your Real Application Clusters design strategy: your goal is to reduce the number of rows for each block and thus the probability of conflicting access. Indicators of very *hot* globally accessed blocks include frequent convert timeouts or consistent read timeouts.

If you see a high proportion of global cache resource waits per transaction, then consider determining which files and blocks are accessed frequently from all nodes. The following describes how to use three views to identify files that are experiencing contention.

Identifying Contended Objects with V$CACHE, V$CACHE_TRANSFER, and V$BH

The V$CACHE, V$CACHE_TRANSFER, and V$BH views show the block classes and blocks that Oracle transfers over the interconnect on a per-object basis. The value for FORCED_WRITES in these views should always be 0 (zero). This is because an LMS process transfers blocks directly to the remote instance's cache. LMS processes ship both the consistent read and the current blocks. In this case, the LMS process starts, but its role is reduced.

With Cache Fusion, the values in the FORCED_READS column indicates the number of times Oracle transferred a current version of particular block from another cache. Therefore, evaluating the values in the FORCED_READS column helps identify tables and indexes that are subject to high cache fusion activity.

Contention for Segment Headers and Free List Blocks

Contention for segment headers can occur when Oracle must read table or index headers while many transactions simultaneously update them. This usually happens when transactions search for blocks with enough free space to hold the data to be inserted or updated. Oracle also updates a segment header if Oracle adds new extents or additional free blocks the table.

New applications that insert a significant amount of data from multiple nodes can become serious performance bottlenecks. This is because Oracle copies the segment header block containing the free lists into another instance's cache. This can result in a single point of contention.

You can significantly improve this situation by creating free list groups for the tables and indexes causing the problem. The advantage of using free list groups is to partition access to segment free lists according to instance. This reduces conflicts between among when the INSERT and DELETE rates are excessive.

Note: Automatic segment-space management in Oracle9*i* eliminates the need to create multiple free lists or free list groups.

Contention for Resources other than Database Blocks

Contention for resources other than database blocks should be infrequent. However, when this occurs, it adversely affects performance. Usually, there are two areas that can exhibit such problems as discussed in the following sections:

- Contention for the Data Dictionary Cache and The Row Cache
- Contention for the Library Cache

Contention for the Data Dictionary Cache and The Row Cache

The use of uncached Oracle sequences or poorly configured space parameters for a table or **tablespace** can cause frequent data dictionary changes. If Oracle frequently reads data dictionary objects and updates them from more than one instance, then Oracle flushes the changes to the redo log and sends the block to the requesting instance. Unfortunately, these objects are often used in recursive, internal transactions while other resources are held. Due to the complexity of these internal data dictionary transactions, this processing can cause serious delays.

These delays increase the values for the *row cache lock* wait count and *time waited* statistics in V$SYSTEM_EVENT. Thus, these events become two of the most waited-for events. The percentage of time waited for row cache locks should never be more than 5% of the total wait time.

Query the V$ROWCACHE view to determine which objects in the row cache might be causing delays. Oracle's response to your query is the name of the row cache object type, such as DC_SEQUENCES or DC_USED_EXTENTS, as well as the GCS requests and GCS conflicts for these objects. If the conflicts exceed 10 to 15 percent of the requests, and if the row cache lock wait time is excessive, then resolve the conflicts by caching sequence numbers, defragmenting the tablespaces, or by tuning the space parameters.

Most frequently, problems occur when Oracle creates sequences without the CACHE option. In these cases, the values for DC_SEQUENCES show high GCS conflict rates. Frequent space management operations due to fragmented tablespaces or inadequate extent sizes can result in cache transfers for DC_USED_EXTENTS and DC_FREE_EXTENTS objects in the data dictionary cache.

Frequent data dictionary changes usually affect data blocks from the data dictionary tables. These blocks normally belong to file number 1. In this situation, you can find copies of these blocks in each instance's buffer cache when you query V$CACHE or V$CACHE_TRANSFER.

> **Note:** Oracle stores rows from the data dictionary in a separate cache. Oracle also uses a different buffer format for these rows than the buffer format Oracle uses in the data buffer cache.

Contention for the Library Cache

Library cache performance problems in Real Application Clusters are rare. They usually manifest themselves as excessive wait times for *library cache pins*. They can result from frequent re-parsing of cursors or from the loading of procedures and packages. Query the DLM columns in the `V$LIBRARYCACHE` view to obtain more detail about this problem.

Cross-instance invalidations of library cache objects should be rare. However, such invalidations can occur if you drop objects referenced by a cursor that is executed in two instances.

Contention Problems Specific to Applications Running on Real Application Clusters

There are some significant contention problems that you should avoid in Real Application Clusters environments. These contention problems result from inserts into index blocks when multiple instances share sequence generators for primary key values. You can minimize these problems by doing the following as described in this section:

- Using Sequence Number Multipliers
- Using the CACHE Clause When Creating Oracle Sequences

Using Sequence Number Multipliers

A sequence number multiplier can prevent instances from inserting new entries into the same index. For example, use a multiplier such as `SEQUENCE_NUMBER` x `INSTANCE_NUMBER` x 1,000,000,000. This multiplier greatly increases the likelihood that a sequence number for an instance is unique.

Using the CACHE Clause When Creating Oracle Sequences

Always use the `CACHE` clause when creating Oracle sequences. Creating sequences without using the `CACHE` clause can create excess overhead. It can also cause synchronization overhead if both instances use the same sequence.

> **See Also:** *Oracle9i SQL Reference* for more information about `SEQUENCE_NUMBER`, `INSTANCE_NUMBER`, and the `CACHE` clause

Resolving Performance Problems in Real Application Clusters-Based Applications

This section explains how to identify and resolve performance problems in Real Application Clusters-based applications. It contains the following topics:

Query Tuning Tips

Query-intensive applications benefit from tuning techniques that maximize the amount of data for each I/O request. Before trying these techniques, monitor performance both before and after implementing them to assess their effectiveness. The techniques are:

Using Large Block Sizes

Use a large block size to increase the number of rows that each operation retrieves. A large block size also reduces the depth of your application's index trees. Your block size should be at least 8K if your database is used primarily for processing queries.

Increasing the Value for DB_FILE_MULTIBLOCK_READ_COUNT

Also set the value for `DB_FILE_MULTIBLOCK_READ_COUNT` to the largest possible value. Doing this improves the speed of full table scans by reducing the number of reads required to scan the table. Note that your system I/O is limited by the block size multiplied by the number of blocks read.

If you use operating system **striping**, then set the stripe size to `DB_FILE_MULTIBLOCK_READ_COUNT` multiplied by the `DB_BLOCK_SIZE` times 2. If your system can differentiate index stripes from table data stripes, then use a stripe size of `DB_BLOCK_SIZE` multiplied by 2 for indexes.

Also remember to:

- Use read-only tablespaces for data and indexes
- Define tablespaces holding temporary segments as type TEMPORARY

Application Tuning Tips

Transaction-based applications generally write more data to disk than other application types. You can use several methods to optimize transaction-based applications. However, you cannot use these techniques on all types of systems. Monitor your application's performance both before and after initiating these methods to make sure it is acceptable.

To improve the ability of the database writer processes (DBWn) to write large amounts of data quickly, use asynchronous I/O. Oracle uses multiple DBWn processes to improve performance.

Note: Not all platforms support asynchronous I/O.

If you have partitioned users by instance and if you have enough space to accommodate the multiple free lists, then use free list groups. This helps your application avoid dynamic data partitioning. You can also manually partition tables by value. If the access is random, then consider using a smaller block size.

In addition to these points, also:

- Be aware of contention on related indexes and sequence generators
- Consider using a multi-tiered architecture to route users to achieve data affinity and to use failover

Diagnosing Performance Problems

If your application is not performing well, then analyze each component of the application to identify which components are causing problems. To do this, check the operating system and GCS statistics for signs of contention or excessive CPU usage as explained under the next heading. Measurable block mode conversions that are excessive can reveal high read/write activity or high CPU requirements by GCS components.

GCS Statistics for Monitoring Contention and CPU Usage

If your application is not performing optimally, then consider examining statistics as described in the following points:

- Use standard tuning techniques by running Statspack or UTLBSTAT and UTLESTAT. Then query the V$SQL view. Examine the statistics from this view and analyze the hit ratios in the shared pool and the buffer cache.
- Examine the dynamic performance table statistics that are created when you run CATCLUST.SQL.
- Use the V$LOCK_ACTIVITY table to monitor block mode conversion rates.
- Use the V$BH table to identify which blocks are being forced written to disk. The V$BH table sums the number of times each block's access modes are downgraded from exclusive to null.
- The V$CACHE_TRANSFER view shows rows from the V$CACHE table where the exclusive-to-null count is non-zero.

Advanced Queuing and Real Application Clusters

Using advanced queuing in Real Application Clusters environments introduces functionality and performance-related issues as described in this section:

- Queue Table Instance Affinity
- Global Cache Service Resource Acquisition
- Advanced Queuing and Queue Table Cache Transfers

Queue Table Instance Affinity

Queue table instance affinity enables you to assign primary and secondary instance properties to queue tables. This allows automatic assignment of queue table ownership when instances shut down and restart. You can evaluate queue table instance affinity by querying the following views:

- DBA_QUEUE_TABLES
- USER_QUEUE_TABLES

Global Cache Service Resource Acquisition

GCS resource acquisition is more expensive than local resource or local enqueue acquisition. If you improperly deploy advanced queuing, then its resource control behavior can adversely affect performance in Real Application Clusters environments. To avoid this, consider implementing the following:

- Increase the number of blocks that are added to a free list when advancing the **high water mark**
- Disable the table locks on queue tables
- Reduce the number of COMMITs

> **See Also:** "Disabling Table Locks for Individual Tables" on page 4-19 for more information on disabling table locks

Advanced Queuing and Queue Table Cache Transfers

In general, cache transfers of queue table data blocks and queue table index blocks can occur under the following circumstances:

- Queues are accessed simultaneously from different instances
- Oracle incorrectly assigns queue table ownership so that a queue monitor schedules a queue from an instance that is different from the instance where the enqueue or dequeue operations are performed
- Oracle must perform space transactions on the queue table

You can reduce the frequency of cache block transfers by:

- Increasing the number of blocks that Oracle adds to a free list when advancing the high water mark
- Using free lists and free list groups for the queue table indexes
- Partitioning applications to access a queue from only one instance
- Partitioning applications to create queue tables for each instance

> **See Also :** *Oracle9i Application Developer's Guide - Advanced Queuing* for general information about using Advanced Queuing

6

Tuning Real Application Clusters and Inter-Instance Performance

This chapter describes **Oracle Real Application Clusters**- and **Cache Fusion**-related statistics and provides procedures that explain how to use them to monitor and tune performance. This chapter also briefly explains how Cache Fusion resolves reader / writer and writer / writer conflicts in Real Application Clusters by describing Cache Fusion's benefits in general terms that apply to most system and application types. The topics in this chapter include:

- How Cache Fusion Produces Current and Consistent Read Blocks
- The Interconnect and Interconnect Protocols for Real Application Clusters
- Performance Expectations of Cache Fusion
- Monitoring Cache Fusion and Inter-Instance Performance
- Cache Fusion and Performance Monitoring Goals
- Statistics for Monitoring Real Application Clusters and Cache Fusion
- Using the V$SYSTEM_EVENT View to Identify Performance Problems

> **See Also:** *Oracle9i Real Application Clusters Concepts* for an overview of Cache Fusion processing

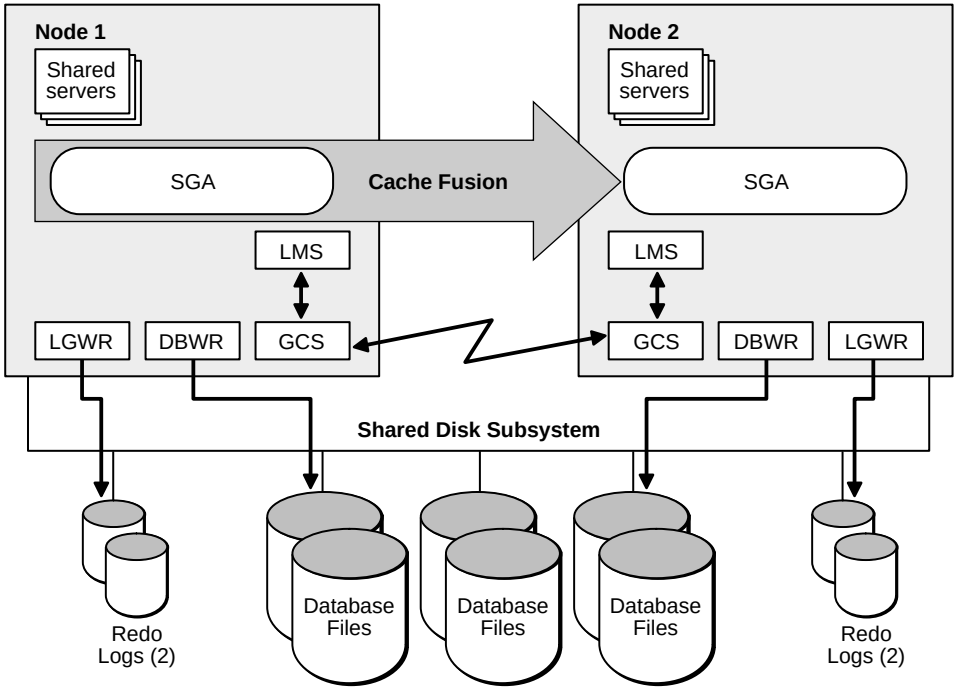
How Cache Fusion Produces Current and Consistent Read Blocks

When one **instance** requests the most current copy of a data block and that copy is in the memory cache of a remote **instance**, Cache Fusion resolves the read / write or write / write conflict using remote memory access, not disk access.

When an instance sends a request for a consistent-read copy or current image of a block to a holding instance, the holding instance logs the changes made to the block and forces a log flush. **Global Cache Service Processes (LMSn)** on the holding instance transmit the requested image of the block, as consistent read blocks or current blocks, directly from the holding instance's buffer cache to the requesting instance's buffer cache across a high speed **interconnect**.

Figure 6–1 illustrates how Cache Fusion enables the buffer cache of one **node** to send data blocks directly to the buffer cache of another node using low latency, high speed **interconnect**s. This reduces the need for expensive disk I / O. Cache Fusion also leverages new interconnect technologies for low latency, user-space based, interprocessor communication. This potentially lowers CPU use by reducing operating system context switches for inter-node messages.

Figure 6–1 Cache Fusion Ships Blocks from Cache to Cache Across the Interconnect



Note: Cache Fusion is always enabled.

Improved Scalability with Cache Fusion

Cache Fusion improves application transaction throughput and scalability by providing:

- Resolution of writer / writer and reader / writer cache coherency conflicts without disk I / O
- Reduced context switches and thus, reduced CPU use due to shorter sequences for round-trip messages
- Significantly reduced CPU use for **User-mode IPC** platforms
- Current and consistent-read block transfers by way of high speed interconnects

Applications with hot spots due to blocks read or modified by multiple instances or benefit the most from Cache Fusion. Packaged applications also scale better as a result of Cache Fusion. Applications in which online transaction processing (OLTP) and reporting functions execute on separate nodes can also take advantage of Cache Fusion. In particular, the more critical OLTP response times benefit significantly from Cache Fusion.

Reporting functions that access data from tables modified by OLTP functions receive consistent read versions of data blocks by way of high speed interconnects. Oracle9*i* release 1 (9.0.1) reduces the **forced disk writes** of data blocks. Performance gains are derived primarily from reduced X-to-S block access mode conversions and the corresponding reduction in disk I / O for X-to-S block access mode conversions.

Furthermore, the instance that was changing the cached data block before it received a read request for that block from another instance did not have to request exclusive access to the block again for subsequent changes. This is because the instance that was changing the block retains the block in exclusive mode after the block is shipped to the reading instance.

With Cache Fusion in Oracle9*i* release 1 (9.0.1), when an instance requests a block in exclusive mode that is currently dirty and in another instance's cache, Oracle still performs an X-to-N block mode conversion (or X-to-S if the block was globally clean). However, this down-convert operation is less expensive in release 1 (9.0.1) in terms of overhead, because the requesting instance receives the block by way of the interconnect. This is significantly faster than the forced disk write processing typical of previous Oracle releases.

Note: All applications achieve performance gains from Cache Fusion. The degree of improvement depends on your operating system, your application's workload, and your overall system configuration.

Block Transfers Using High Speed Interconnects

Because Cache Fusion exploits high speed IPCs, Real Application Clusters benefits from the performance gains of the latest technologies for low latency communication across **cluster** interconnects. You can expect even greater performance gains if you use more efficient protocols, such as **Virtual Interface Architecture (VIA)** and user-mode IPCs.

Cache Fusion reduces CPU use by taking advantage of user-mode IPCs, also known as *memory-mapped IPCs*, for UNIX, Windows NT, and Windows 2000 platforms. If the appropriate hardware support is available, then **operating system context switches** are minimized beyond the basic reductions achieved with Cache Fusion alone. This also eliminates costly data copying and system calls.

If efficiently implemented by your hardware, then user-mode IPCs can reduce CPU use. This is because user processes can communicate without using the operating system kernel in user-mode IPC architectures. In other words, user processes do not have to switch from user execution mode to kernel execution mode.

Elimination of I/O for Forced Disk Writes of Blocks

Cache Fusion practically eliminates disk I/O for data and undo segment blocks by transmitting current block mode versions and consistent-read blocks directly from one instance's buffer cache to another. This can reduce the latency required to resolve writer/writer and reader/writer conflicts by as much as 90 percent.

Cache Fusion resolves concurrency, as mentioned, without disk I/O. Cache Fusion expends only one tenth of the processing effort that was required by disk-based parallel cache management. To do this, Cache Fusion only incurs overhead for:

- Pinning a current block, logging the changes to the block, forcing a log flush, and sending the block

Or when requesting a consistent read version:

- Processing the request and constructing a consistent-read copy of the requested block in memory and transferring it to the requesting instance

On some platforms this can take less than one millisecond.

Partitioning Data To Further Reduce Hot Spots Due to Blocks Modified by Multiple Instances

You can accurately assess hot spots due to blocks modified by multiple instances by evaluating global cache resource busy and **buffer busy due to global cache** statistics using V$SYSTEM_EVENT as described in Table 5–3 on page 5-14. If your application has hot spots, consider using one of the partitioning techniques in Chapter 3 to further reduce the cost of writer / writer conflicts.

The Interconnect and Interconnect Protocols for Real Application Clusters

The primary components affecting Cache Fusion performance are the interconnect and the inter-node communication protocols. The interconnect bandwidth, its latency, and the efficiency of the IPC protocol determine the speed with which Cache Fusion processes consistent-read block requests.

> **See Also:** *Oracle9i Real Application Clusters Installation and Configuration* for a list of the supported interconnect protocols

Influencing Interconnect Processing

Once your interconnect is operative, you cannot significantly influence its performance. However, you can influence an interconnect protocol's efficiency by adjusting the IPC buffer sizes.

> **See Also:** For more information, consult your vendor-specific interconnect documentation

Performance Expectations of Cache Fusion

Cache Fusion performance levels can vary in terms of latency and throughput from application to application. Performance is further influenced by the type and mixture of transactions that your system processes.

The performance gains from Cache Fusion also vary with each workload. The hardware, the interconnect protocol specifications, and the operating system resource use also affect performance.

If your application did not demonstrate a significant amount of consistent-read or write / write contention prior to Cache Fusion, then your performance with Cache Fusion will likely remain unchanged. However, if your application experienced numerous block mode conversions and heavy disk I / O as a result of consistent-read or writer / writer conflicts, your performance with Cache Fusion should improve significantly.

The following section, "Monitoring Cache Fusion and Inter-Instance Performance", describes how to evaluate Cache Fusion performance in more detail.

Monitoring Cache Fusion and Inter-Instance Performance

This section describes how to obtain and analyze Real Application Clusters and Cache Fusion statistics to monitor inter-instance performance. Topics in this section include:

- Cache Fusion and Performance Monitoring Goals
- Statistics for Monitoring Real Application Clusters and Cache Fusion
- Using the V$SYSTEM_EVENT View to Identify Performance Problems

Cache Fusion and Performance Monitoring Goals

The main goal of monitoring Cache Fusion and Real Application Clusters performance is to determine the cost of global processing and to quantify the resources required to maintain coherency and synchronize the instances. Do this by analyzing the performance statistics from several views as described in the following sections. Use these monitoring procedures on an on-going basis to observe processing trends and to optimize processing.

Many statistics measure the work done by different components of the database kernel, such as the cache layer, the transaction layer, or the I / O layer. Moreover, timed statistics allow you to accurately determine the amount of time spent processing certain requests or the amount of time waited for specific events. From these statistics, you can derive work rates, wait times, and efficiency ratios.

See Also:

- Chapter 5 for additional suggestions on which statistics to collect and how to use them to compute performance ratios
- Chapter 7 for more information on statistics gathered by **Oracle Enterprise Manager**, and for information on using Oracle Enterprise Manager to monitor Real Application Clusters environments

Statistics for Monitoring Real Application Clusters and Cache Fusion

Oracle collects Cache Fusion-related performance statistics from the buffer cache and **Global Cache Service (GCS)** layers. Oracle also collects general Real Application Clusters statistics for block requests and block mode conversion waits. You can use several views to examine these statistics.

Maintaining an adequate history of system performance helps identify trends as these statistics change. This helps identify problems that contribute to increased response times and reduced throughput. A history of performance trends is also helpful in identifying workload changes and peak processing requirements.

Procedures in this section use statistics grouped according to the following topics:

- Analyzing Global Cache and Cache Fusion Statistics
- Analyzing Global Enqueue Statistics
- Analyzing GES Resource, Message, and Memory Resource Statistics
- GES Message Statistics Processing
- Analyzing Latch Statistics in Real Application Clusters

As described in Chapter 5, consider maintaining statistics from the V$SYSSTAT view and the V$SYSTEM_EVENT view on a per second and per transaction basis. This reveals a general profile of the workload. Relevant observations from these views are:

- Requests or counts per transaction, for example, global cache current blocks served per transaction
- Wait times or elapsed times per transaction, for example, time per transaction for waits for busy buffers, and block mode convert times per transaction
- Requests or counts per second

- Average times per request, for example, average time to receive a consistent read buffer from another instance

By maintaining these statistics, you can accurately estimate the effect of an increasing cost for a certain type of operation on transaction response times. Major increases in work rates or average delays also contribute to identifying capacity issues.

You must set the parameter TIMED_STATISTICS to true to make Oracle collect statistics for most views discussed in the procedures in this section. The timed statistics from views discussed in this chapter are displayed in units of 1/100ths of a second.

Creating Real Application Clusters Data Dictionary Views with CATCLUST.SQL

You must run CATCLUST.SQL to create Real Application Cluster-related views and tables for storing and viewing statistics. To run this script, you must have SYSDBA privileges.

CATCLUST.SQL creates the standard V$ dynamic views, as well as:

- GV$CACHE
- GV$CACHE_TRANSFER
- GV$CLASS_CACHE_TRANSFER
- GV$FILE_CACHE_TRANSFER
- GV$ROWCACHE
- GV$LIBRARYCACHE

Even though the shared disk architecture eliminates forced disk writes, V$FILE_CACHE_TRANSFER, V$CLASS_CACHE_TRANSFER, and V$CACHE_TRANSFER still show the number of block mode conversions per block class or object. However, as mentioned, the FORCED_WRITES column will be 0 (zero). This indicates that, for example, an X-to-N conversion resulted in a cache block transfer without incurring forced disk writes.

> **See Also:** *Oracle9i Database Reference* for more information on dynamic performance views

Global Dynamic Performance Views

Each active instance has its own set of instance-specific views. You can also query global dynamic performance views to retrieve the V$ view information from all qualified instances. Global dynamic performance views' names are prefixed with *GV$*. A global fixed view is available for all dynamic performance views except:

- V$ROLLNAME
- V$CACHE_LOCK
- V$LOCK_ACTIVITY
- V$LOCKS_WITH_COLLISIONS

The global view contains all columns from the instance-specific view, with an additional column, INST_ID of datatype INTEGER. This column displays the **instance number** from which the associated V$ information was obtained. You can use the INST_ID column as a filter to retrieve V$ information from a subset of available instances. For example, the query retrieves information from the V$ views on instances 2 and 5:

```
SELECT * FROM GV$LOCK WHERE INST_ID = 2 or INST_ID = 5;
```

Each global view contains a GLOBAL hint that creates a query executed in parallel to retrieve the contents of the local views on each instance.

If you have reached the limit of PARALLEL_MAX_SERVERS on an instance and you attempt to query a GV$ view, then Oracle spawns one additional **parallel execution** server process for this purpose. The extra process is not available for parallel operations other than GV$ queries.

> **Note:** If PARALLEL_MAX_SERVERS is set to zero for an instance, then additional parallel execution server processes do not spawn to accommodate a GV$ query.

If you have reached the limit of PARALLEL_MAX_SERVERS on an instance and issue multiple GV$ queries, then all but the first query will fail. In most parallel queries, if a server process could not be allocated it would result in either an error or a sequential execution of the query by the query coordinator.

See Also:

- "Flexible Parallelism within Real Application Clusters Environments" on page 2-6
- *Oracle9i Database Reference* for restrictions on GV$ views and complete descriptions of all related parameters and V$ dynamic performance views

Analyzing Global Cache and Cache Fusion Statistics

Oracle collects global cache statistics at the buffer cache layer within an instance. These statistics include counts and timings of requests for global resources.

Requests for global resources for data blocks originate in the buffer cache of the requesting instance. Before a request enters the GCS request queue, Oracle allocates data structures in the **System Global Area (SGA)** to track the state of the request. These structures are called *resources*.

To monitor global cache statistics, query the V$SYSSTAT view and analyze its output as described in the following procedures and equations.

Procedures for Monitoring Global Cache Statistics

Complete the following steps to analyze global cache statistics.

1. Use the following syntax to query V$SYSSTAT:

```
SELECT NAME,VALUE FROM V$SYSSTAT WHERE NAME LIKE '%global cache%';
```

Oracle responds with output similar to:

```
NAME                                                  VALUE
----------------------------------------- ----------
global cache blocks corrupt                           0
global cache blocks lost                              0
global cache claim blocks lost                        0
global cache convert time                        287709
global cache convert timeouts                         0
global cache converts                            137879
global cache cr block build time                   4328
global cache cr block flush time                   9565
global cache cr block receive time               742421
global cache cr block send time                   10119
global cache cr blocks received                  448301
global cache cr blocks served                    442322
```

```
global cache current block flush time          10944
global cache current block pin time            27318
global cache current block receive time       254702
global cache current block send time            2819
global cache current blocks received          132248
global cache current blocks served            130538
global cache defers                             4836
global cache freelist waits                        0
global cache get time                          15613
global cache gets                               9178
global cache prepare failures                      0
23 rows selected.
```

2. Calculate the average latency of a consistent block request, in other words, its round-trip time, as:

$$\frac{\textit{global cache cr block receive time}}{\textit{global cache cr blocks received}}$$

   The result, which should typically be about 15 milliseconds depending on your system configuration and volume, is the average latency of a consistent-read request round-trip from the requesting instance to the holding instance and back to the requesting instance. If your CPU has limited idle time and your system typically processes long-running queries, then the latency may be higher. However, it is possible to have an average latency of less than one millisecond with **User-mode IPC**.

   Request latency can also be influenced by a high value for the `DB_MULTI_BLOCK_READ_COUNT` parameter. This is because a requesting process can issue more than one request for a block depending on the setting of this parameter. Correspondingly, the requesting process may wait longer.

3. For a high number of incoming requests, especially in report-intensive applications, or if there are multiple nodes from which requests are dispatched to an LMS process, the round-trip time can increase because LMS' service time increases. To determine whether the length of the delay is caused by LMS

overhead, compute the average service time per request using the following equation:

$$\frac{\textit{global cache cr [queue + build + flush + send] time}}{\textit{global cache cr blocks served}}$$

Over a period of time during peak processing intervals, track the average LMS service time per request and the total round-trip time per request as presented in this step.

To determine which part of the service time correlates most with the total service time, derive the time waited to queue the consistent-read request, the time needed to build the consistent-read block, the time waited for a log flush and the time spent to send the completed request using the following three equations.

Time needed to build the consistent-read block:

$$\frac{\textit{global cache cr block build time}}{\textit{global cache cr blocks served}}$$

Time waited for a log flush:

$$\frac{\textit{global cache cr block flush time}}{\textit{global cache cr blocks served}}$$

Time spent to send the completed request:

$$\frac{\textit{global cache cr block send time}}{\textit{global cache cr blocks served}}$$

By calculating these averages, you can account for almost all the processing steps of a consistent read block request. The remaining difference between the total round-trip time and the LMS service time per request is the queuing time

for the **Global Enqueue Service Daemon (LMD)** to dispatch the request to the LMS process and the processing time for LMS and network IPC time.

4. Calculate the average latencies for a request for a current block:

$$\frac{\textit{global cache current block receive time}}{\textit{global cache current blocks received}}$$

The following equation is for receiving a current block:

$$\frac{\textit{global cache current block [pin + flush + send] time}}{\textit{global cache current blocks served}}$$

5. Calculate which part of the time is spent to serve a request for a current block:

The service time for a current block request encompasses the pin time and the send time. The next two equations help you determine processing times for the various steps of a cache transfer.

Determining pin time:

$$\frac{\textit{global cache current block pin time}}{\textit{global cache current blocks served}}$$

Determining send time:

$$\frac{\textit{global cache current block send time}}{\textit{global cache current blocks served}}$$

6. Calculate the average convert times and average get times using one of these formulas:

$$\frac{\textit{global cache convert time}}{\textit{global cache converts}} \quad \textit{or} \quad \frac{\textit{global cache get time}}{\textit{global cache gets}}$$

When analyzing the results from this step, consider the following points:

- High convert times can indicate excessive global concurrency.
- A large number of global cache gets, global cache converts, and rapid increases in average convert or get times indicates excessive contention for GCS operations
- For Global Enqueue Service (GES) operations, refer to "Analyzing Global Enqueue Statistics" on page 6-16
- Latencies for resource operations can be excessive due to overall system workload or system problems.

> **Note:** A reasonable value for a cache get is 20 to 30 milliseconds while converts should average 10 to 20 milliseconds.

Oracle increments global cache gets when a new resource is opened. A convert is counted when there is already an open resource and Oracle converts the resource to another mode.

Therefore, the elapsed time for a get includes the allocation and initialization of new resources. If the average cache get or average convert times are excessive, then your system may be experiencing timeouts.

If the global cache convert times or global cache get times are excessive, then refer to statistics in the `V$SYSTEM_EVENT` view to identify events with a high value for `TIME_WAITED` statistics.

7. Analyze block mode conversion timeouts by examining the value for **global cache convert timeouts**. If your `V$SYSSTAT` output shows a value other than zero for this statistic, then check your system for congestion or high contention. In general, convert timeouts should not occur; their existence indicates serious performance problems.

8. Analyze the global cache consistent-read timeouts by examining the value for this statistic in your V$SYSSTAT output. Oracle increments this statistic after the system waits too long for the completion of a consistent-read request. If this statistic shows a value other than zero, then too much time has elapsed after the initiation of a consistent-read request and a timeout has occurred. If this happens, then you also usually find that the average time for consistent-read request completions has increased. If you have timeouts and the latency is high, then your system may have an excessive workload or there may be excessive contention for data blocks. Timeouts might also indicate IPC or network problems.

Analyzing Global Enqueue Statistics

Global enqueue statistics provide latencies and counts for **Global Enqueue Service (GES)** activity. Oracle collects global enqueue statistics from the GES API layer. All Oracle clients to the GES make their requests to the GES through this layer. Thus, global resource statistics include global enqueue requests originating from all layers of the kernel while global cache statistics relate to buffer cache activity.

Use the procedures in this section to monitor data from the V$SYSSTAT view to derive GES latencies, counts, and averages. This helps estimate the Real Application Clusters workload generated by an instance.

Procedures for Analyzing Global Enqueue Statistics

Use the following procedures to view and analyze statistics from the V$SYSSTAT view for global enqueue processing.

1. Use this syntax to query V$SYSSTAT:

```
SELECT NAME, VALUE FROM V$SYSSTAT WHERE NAME LIKE '%global lock%';
```

Oracle responds with output similar to:

```
NAME                                                             VALUE
---------------------------------------------------------------- ----------
global lock sync gets                                               703
global lock async gets                                            12748
global lock get time                                               1071
global lock sync converts                                           303
global lock async converts                                           41
global lock convert time                                             93
global lock releases                                                573
7 rows selected.
```

Use your V$SYSSTAT output to perform the calculations and analyses described in procedures 2 through 5.

2. Calculate the average global enqueue get time using this formula:

$$\frac{\textit{global lock get time}}{\textit{(global lock sync gets + global lock async gets)}}$$

If the result is more than 20 or 30 milliseconds, then query the TIME_WAITED column in the V$SYSTEM_EVENT view using the descending (DESC) keyword to identify which resource events are waited for most frequently using this query:

```
SELECT TIME_WAITED, AVERAGE_WAIT
FROM V$SYSTEM_EVENT
ORDER BY TIME_WAITED DESC;
```

Oracle increments global lock gets when Oracle opens a new global enqueue for a resource. A convert is counted when there is already an open global enqueue on a resource and Oracle converts the global enqueue access mode to another mode.

Thus, the elapsed time for a get includes the allocation and initialization of a new global enqueue. If the average global enqueue get (**global cache get time**) or average global enqueue (from the above formula) conversion times are excessive, then your system may be experiencing timeouts.

If the global enqueue conversion times or global enqueue get times are high, then refer to statistics in the V$SYSTEM_EVENT view to identify events with a high value for TIME_WAITED statistics.

3. Calculate the average **global lock convert time** using this formula:

$$\frac{\textit{global lock convert time}}{\textit{(global lock sync converts + global lock async converts)}}$$

If the result is more than 20 milliseconds, then query the TIME_WAITED column in the V$SYSTEM_EVENT view using the DESC keyword to identify the event causing the delay.

4. As mentioned, global enqueue statistics apply only to operations that do not involve data blocks. To determine which types of resources may be causing performance problems, divide the global enqueue get and global enqueue conversion statistics into two categories:

   - Synchronous Operations

     Synchronous global enqueue gets include, for example, global lock sync gets. To determine the proportion of the time required for synchronous global enqueue gets, divide the global lock gets time or global lock convert time by the corresponding number of synchronous operations.

   - Asynchronous Operations

     Asynchronous global enqueue operations include, for example, **global lock async gets**. These are typically global enqueue operations from non-blocking process to synchronize inter-instance activity. You can derive the proportion of the total time using the same calculation as you used for synchronous operations. In this way, you can determine the proportion of work and the cost of global enqueue requests.

     Normally, if the proportion of global enqueue requests other than global cache requests dominates the cost for all global resource operations, the V$SYSTEM_EVENT view shows high wait times for row cache locks, enqueues or library cache pins.

5. Analyze the V$LIBRARYCACHE and V$ROWCACHE views to observe GES activity on local resources. These views have GES-specific columns that identify GES resource use. Analyze these views for GES activity if you have frequent and extended waits for library cache pins, enqueues, or **DFS Lock Handles**.

Analyzing GES Resource, Message, and Memory Resource Statistics

Use GES resource and message statistics to monitor GES latency and workloads. These statistics appear in the V$GES_CONVERT_LOCAL and V$GES_CONVERT_REMOTE views.

These views record average convert times, count information, and timed statistics for global enqueue requests. The V$GES_CONVERT_LOCAL view shows statistics for local GES enqueue operations. The V$GES_CONVERT_REMOTE view shows values for remote GES enqueue conversions. These views display the average convert times in 100ths of a second.

How GES Workloads Affect Performance

The Global Enqueue Service (GES) manages all the non-Cache Fusion intra- and inter-instance resource operations. High GES workload request rates can adversely affect performance. The GES performs local enqueue resource operations entirely within the local node, or in other words, without sending messages. Remote enqueue resource operations require sending messages to and waiting for responses from other nodes. Most down-converts, however, are local operations for the GES.

The following procedures for analyzing GES resource and message statistics appear in two groups. The first group of procedures explains how to monitor GES resources. The second group explains how to monitor message statistics.

Procedures for Analyzing GES Resource Statistics

Use the following procedures to obtain and analyze statistics from the V$GES_CONVERT_LOCAL and V$GES_CONVERT_REMOTE views for GES resource processing.

> **Note:** The $GES_CONVERT_LOCAL and V$GES_CONVERT_REMOTE views still include rows for block mode conversions from and to SSX, although SSX no longer exists. Therefore, the value in the CONVERT_COUNT and AVERAGE_CONVERT_TIME columns for SSX conversions is always zero.

You must enable event 29700 to populate the V$GES_CONVERT_LOCAL and V$GES_CONVERT_REMOTE views. Do this by entering the following syntax in your **initialization parameter file**:

```
EVENT="29700 TRACE NAME CONTEXT FOREVER"
```

1. Use this syntax to query the V$GES_CONVERT_LOCAL view:

```
SELECT CONVERT_TYPE,
AVERAGE_CONVERT_TIME,
CONVERT_COUNT
FROM V$GES_CONVERT_LOCAL;
```

Oracle responds with output similar to:

```
CONVERT_TYPE                         AVERAGE_CONVERT_TIME CONVERT_COUNT
------------------------------------ -------------------- -------------
NULL -> SS                                              0             0
NULL -> SX                                              0             0
```

```
NULL -> S                                                1          146
NULL -> SSX                                              0            0
NULL -> X                                                1           92
SS   -> SX                                               0            0
SS   -> S                                                0            0
SS   -> SSX                                              0            0
SS   -> X                                                0            0
SX   -> S                                                0            0
SX   -> SSX                                              0            0
SX   -> X                                                0            0
S    -> SX                                               0            0
S    -> SSX                                              0            0
S    -> X                                                3           46
SSX  -> X                                                0            0
16 rows selected.
```

2. Use this syntax to query the V$GES_CONVERT_REMOTE view:

   ```
   SELECT * FROM V$GES_CONVERT_REMOTE;
   ```

   Oracle responds with output that is identical in format to the output from the V$GES_CONVERT_LOCAL view.

   Use output from the V$GES_CONVERT_LOCAL and V$GES_CONVERT_REMOTE views to perform the calculation described in the following procedure that performs a join over the two views.

3. Calculate the ratio of local-to-remote global enqueue resource operations using this query:

   ```
   SELECT
    r.CONVERT_TYPE,
    r.AVERAGE_CONVERT_TIME,
    l.AVERAGE_CONVERT_TIME,
    r.CONVERT_COUNT,
    l.CONVERT_COUNT,
   FROM V$GES_CONVERT_LOCAL l, V$GES_CONVERT_REMOTE r
   WHERE r.convert_count <> 0 OR l.convert_count <> 0
   GROUP BY r.CONVERT_TYPE;
   ```

4. It is useful to maintain a history of workloads and latencies for block mode conversions. Changes in resource requests per transaction without increases in average latencies usually result from changing application workload patterns.

   The deterioration of both request rates and latencies usually indicates an increased rate of resource conflicts or an increased workload due to message

latencies, system problems, or timeouts. If the LMD processes show excessive CPU consumption, or if consumption is greater than 20 percent of the CPU while overall system resource consumption is normal, then consider binding the LMD process to a specific processor if the system has more than one CPU. The instance starts at least one LMS process; the instance sometimes starts more than one LMS process depending on the availability of system resources.

If latencies increase, then examine CPU data and other operating system statistics that you can obtain using utilities such as *sar, vmstat* and *netstat* on UNIX or *Perfmon* on Windows NT and Windows 2000.

5. Derive an estimate of CPU busy time for LMD from the V$SYSTEM_EVENT view.

   For a quick estimate of the CPU time spent by LMD, transform the wait time event for LMD presented in the V$SYSTEM_EVENT view. To do this, look for the event name *ges remote messages* that represents the time that the LMD process is idle. The TIME_WAITED column contains the accumulated idle time for LMD in units of hundredths of a second.

   To derive the busy time, divide the value for TIME_WAITED by the length of the measurement interval after normalizing it to seconds. In other words, a value of 17222 centiseconds is 172.22 seconds. The result is the idle time of the LMD process, or the percentage of idle time. Subtract that value from 1 and the result is the busy time for the LMD process. This is a fairly accurate estimate when compared with operating system utilities that provide information about CPU utilization per process.

   Note: You should have beginning and ending snapshots to make accurate calculations.

GES Message Statistics Processing

The GES sends messages for both Global Cache and Global Enqueue Services either directly or with flow control. For both methods, the GES attaches a marker, known as a *ticket*, to each message. The allotment of tickets for each GES is limited. However, the GES can re-use tickets indefinitely.

The LMS process, with the LMD process, manages flow-controlled messaging. LMS sends messages to remote instances, and it does this until no more tickets are available. When the GES runs out of tickets, messages must wait in a flow control

queue until outstanding messages have been acknowledged and more tickets are available.

The rationing of tickets prevents one node from sending an excessive amount of messages to another node during periods of heavy inter-instance communication. This also prevents one node with heavy remote consistent-read block requirements from assuming control of messaging resources throughout a cluster at the expense of other, less-busy nodes.

The V$GES_STATISTICS view contains the following statistics about message activity:

- **messages sent directly**
- **messages flow controlled**
- **messages sent indirectly**
- **messages received**
- **flow control messages sent**
- **flow control messages received**

Procedure for Analyzing GES Message Statistics

Use the following procedure to obtain and analyze message statistics in the V$GES_STATISTICS view.

1. Use this syntax to query the V$GES_STATISTICS view:

```
SELECT * FROM V$GES_STATISTICS;
```

Oracle responds with output similar to:

```
STATISTIC# NAME                                             VALUE
---------- ------------------------------------------------ ----------
         0 messages sent directly                            140019
         1 messages flow controlled                            1211
         2 messages sent indirectly                            9485
         3 messages received                                 155287
         4 flow control messages sent                             0
         5 flow control messages received                         0
         6 dynamically allocated enqueues                         0
         7 dynamically allocated resources                        0
         8 gcs msgs received                                 154079
         9 gcs msgs process time(ms)                         192866
        10 ges msgs received                                   1198
```

```
        11 ges msgs process time(ms)                          355
        12 msgs causing lmd to send msgs                      256
        13 lmd msg send time(ms)                                0
        14 gcs side channel msgs actual                      1304
        15 gcs side channel msgs logical                   130400
        16 gcs pings refused                                   68
        17 gcs writes refused                                   3
        18 gcs error msgs                                       0
        19 gcs out-of-order msgs                               30
        20 gcs immediate (null) converts                     1859
        21 gcs immediate cr (null) converts                     5
        22 gcs immediate (compatible) converts                 38
        23 gcs immediate cr (compatible) converts               5
        24 gcs blocked converts                             49570
        25 gcs queued converts                                159
        26 gcs blocked cr converts                         124860
        27 gcs compatible basts                                 0
        28 gcs compatible cr basts                              0
        29 gcs cr basts to PIs                                  0
        30 dynamically allocated gcs resources                  0
        31 dynamically allocated gcs shadows                    0
        32 gcs recovery claim msgs actual                       0
        33 gcs recovery claim msgs logical                      0
        34 gcs write request msgs                              17
        35 gcs flush pi msgs                                    7
        36 gcs write notification msgs                          0
37 rows selected.
```

Note: Oracle support may request information from your V$GES_STATISTICS output for debugging.

Analyzing Block Mode Conversions by Type

This section describes how to analyze output from three views to quantify block mode conversions by type. The tasks and the views discussed in this section are:

- Using the V$LOCK_ACTIVITY View to Analyze Block Mode Conversions
- Using the V$CLASS_CACHE_TRANSFER View to Identify Block Mode Conversions by Block Class
- Using the V$CACHE_TRANSFER View to Identify Hot Objects

Using the V$LOCK_ACTIVITY View to Analyze Block Mode Conversions

The V$LOCK_ACTIVITY view summarizes how many block mode up- and down-converts have occurred during an instance's lifetime. X-to-N or X-to-S down-converts denote the number of times a block mode was down-converted because another instance attempted to modify a block that is currently held in exclusive mode on another instance.

Using the V$CLASS_CACHE_TRANSFER View to Identify Block Mode Conversions by Block Class

The V$CLASS_CACHE_TRANSFER view summarizes block mode conversion activity.

- Data blocks
- Segment headers
- Extent headers
- Undo blocks

With a shared disk architecture, the FORCED_WRITES column is always 0 (zero) because X-to-N or X-to-S down-converts do not result in forced disk writes.

Using the V$CACHE_TRANSFER View to Identify Hot Objects

The V$CACHE_TRANSFER view helps identify *hot* blocks and *hot* objects.

All three views provide different levels of detail. You can monitor the V$LOCK_ACTIVITY view to generate an overall Real Application Clusters workload profile. Use information from the V$LOCK_ACTIVITY view to record the rate at which block mode conversions occur.

For more details, use the V$CLASS_CACHE_TRANSFER view to identify the type of block on which block mode conversions are occurring. Once you have identified the class, use the V$CACHE_TRANSFER view to obtain details about a particular table or index and the file and block numbers on which there is significant block mode conversion activity.

If your response time or throughput requirements are no longer being met, then examine the V$LOCK_ACTIVITY, V$CLASS_CACHE_TRANSFER, V$CACHE, V$CACHE_TRANSFER or V$FILE_CACHE_TRANSFER views. In addition, you might also examine:

- V$SYSSTAT to identify an increase in resource requests per transaction

- V$SYSSTEM_EVENT to identify longer waits for global cache resources or consistent read server requests per transaction
- Global and local work done as described in Chapter 5 to see if there is a noticeable change in performance percentages

In summary, a change in your application profile and work rates typically warrants a detailed analysis using the previously-mentioned views. Apart from diagnosing performance problems of existing applications, these views are also useful when developing applications or when deciding on a partitioning strategy.

Analyzing Latch Statistics in Real Application Clusters

Latches are low-level locking mechanisms that protect SGA data structures. You can use the V$LATCH and V$LATCH_MISSES views to monitor latch contention within the GCS. These views show information about a particular latch, its statistics, and the location in the code from where the latch is acquired.

For normal operations, the value of latch statistics is limited. In some cases, multiple latches can improve performance to a limited degree for certain layers. Excessive latch contention degrades performance and is often the result of one or both of the following:

- Higher level performance issues or a poorly tuned system
- Oracle internal inefficiencies or performance bugs

Use the following procedures to identify latch contention. However, Oracle does not recommend that you monitor these statistics on a regular basis and derive conclusions solely on the basis of latching issues.

In the majority of cases, latch contention will not be your actual performance problem. On the other hand, record information from these procedures if the TIME_WAITED value for the *latch free* wait event is excessive and if it ranks among the events that accrue the largest times as indicated by the V$SYSTEM_EVENT view. In addition, gathering this information might be useful to Oracle Support or Oracle Development.

Procedures for Analyzing Latch Statistics

Use the following procedures to analyze latch, Real Application Clusters, and GCS- and GES-related statistics.

1. Query the V$LATCH view using this syntax:

```
SELECT NAME, GETS, MISSES, SLEEPS FROM V$LATCH;
```

2. If the output reveals a high ratio of sleeps to misses (usually, a ratio above 1 indicates performance issues), then attempt to determine where the sleeps occur. To do this, execute this query on the V$LATCH_MISSES view:

```
SELECT PARENT_NAME, WHERE, SLEEP_COUNT
FROM V$LATCH_MISSES
ORDER BY SLEEP_COUNT DESC;
```

Oracle responds with output similar to:

```
PARENT_NAME                    WHERE                      SLEEP_COUNT
------------------             --------------------       ------------
ges resource hash list         kjrrmas1: lookup master n  39392
cache buffers chains           kcbgtcr: kslbegin          27738
library cache                  kglhdgn: child:            15408
shared pool                    kghfnd: min scan           6876
cache buffers chains           kcbrls: kslbegin           2124
shared pool                    kghalo                     1667
ges process parent             kjucll: delete lock from   1464
7 rows selected.
```

Use your V$LATCH and V$LATCH_MISSES output to perform the following procedures.

3. Calculate the ratio of gets to misses using your V$LATCH output from step 1 in this section using the following formula:

$$\frac{gets}{misses}$$

Excessive numbers for misses usually indicate contention for the same resources. Acceptable ratios range from 90 to 95%.

4. Analyze the ratio of sleeps to misses using your V$LATCH_MISSES output from step 1 in this section. This ratio determines how often a process sleeps when it cannot immediately get a latch.

A ratio of 2 means that for each miss, a process attempts to get a latch twice before acquiring it. A high number of sleeps-to-misses usually indicates process

scheduling delays or high operating system workloads. It can also indicate internal inefficiencies or high concurrency on one resource. For example, when many resources are opened simultaneously, then processes might have to wait for a resource latch.

In the V$LATCH_MISSES view, the WHERE column shows the function in which the latch is acquired. This information is useful in determining internal performance problems. Usually, the latch slept on for long periods shows up in the V$SESSION_WAIT or V$SYSTEM_EVENT views under the 'latch free' wait event category.

The next section describes using the V$SYSTEM_EVENT view in more detail.

Using the V$SYSTEM_EVENT View to Identify Performance Problems

Data about Cache Fusion and Real Application Clusters events appears in the V$SYSTEM_EVENT view. To identify events for which processes have waited the longest, query the V$SYSTEM_EVENT view on the TIME_WAITED column using the descend (DESC) keyword. The TIME_WAITED column shows the total wait time for each system event listed.

By generating an ordered list of wait events, you can easily locate performance bottlenecks. Each COUNT represents a voluntary context switch. The TIME_WAIT value is the cumulative time that processes waited for particular system events. The values in the TOTAL_TIMEOUT and AVERAGE_WAIT columns provide additional information about system efficiency.

Oracle Corporation recommends dividing the sum of values from the TOTAL_WAITS and TIME_WAITED columns by the number of transactions, as outlined in Chapter 5. Transactions in this sense can be defined as business transactions, for example, insurance quotes, order entry, and so on. Or you can define them on the basis of *user commits* or *executions*, depending on your perspective.

The goal is to estimate which event type contributes the most to transaction response times, because in general:

$$\frac{\textit{response time}}{\textit{number of transactions}} = \frac{\textit{CPU time}}{\textit{number of transactions}} + \frac{\textit{wait time}}{\textit{number of transactions}}$$

By this rationale, the total wait time can be divided into subcomponents of the wait time as shown in the following equations where *tm* is *time waited*:

$$\frac{\textit{total wait time}}{\textit{number of transactions}} = \frac{\textit{(db file sequential read tm)}}{\textit{number of transactions}} + \frac{\textit{(global cache cr request tm)}}{\textit{number of transactions}} + \ldots$$

It is also useful to derive the total wait time by adding the individual events and observing the percentages that are spent waiting for each event. This enables you to derive the major cost factors for transaction response times. Reducing the time for the largest proportion of the waits has the most significant effect on response time.

Real Application Clusters Events in V$SYSTEM_EVENT

The following events appearing in the V$SYSTEM_EVENT output represent waits for Real Application Clusters events:

- global cache cr request
- library cache pin
- buffer busy due to global cache
- global cache busy
- global cache open x
- global cache open s
- global cache null to x
- global cache s to x
- global cache null to s

Events Related to Server Coordination Resources

You can monitor other events in addition to those listed under the previous heading because performance problems can be related to server coordination processing within Real Application Clusters. These events are:

- Row cache locks
- Enqueues
- Library cache pins
- DFS lock handle

General Observations for Tuning Inter-Instance Performance

If the time waited for global cache events is high relative to other types of waits, then look for increased latencies, contention, or excessive system workloads. Do this using V$SYSSTAT statistics and operating system performance monitors. Excessive global cache busy or buffer busy waits indicates increased contention in the buffer cache.

> **Note:** In OLTP systems with data block address locking and a high degree of contention, it is not unusual when the global cache wait events represent a high proportion of the sum of the total time waited.

If excessive wait time is used by waits for non-buffer cache resources as shown by statistics in the rows *row cache lock*, *enqueues*, and *library cache pin*, then monitor the V$ROWCACHE and V$LIBRARYCACHE views for Real Application Cluster-related issues. Specifically, observe the values in the DLM columns of each of these views.

Real Application Clusters problems commonly arise from poorly managed space parameters or sequence numbers that are not cached. In such cases, processes wait for row cache locks and enqueues and the V$ROWCACHE view shows excessive conflicts for certain dictionary caches.

Part IV

Using Oracle Enterprise Manager to Monitor and Tune Real Application Clusters Databases

Part Four contains information about using **Oracle Enterprise Manager** to monitor and tune the performance of Real Application Cluster databases. The chapter in Part Four is:

- Chapter 7, "Monitoring Performance with Oracle Performance Manager"

7

Monitoring Performance with Oracle Performance Manager

This chapter presents the **Oracle Performance Manager** performance and tuning charts for **Oracle Real Application Clusters**. You must install and configure Oracle Performance Manager to display the charts. This chapter describes only Oracle Performance Manager features specific to Real Application Clusters. Use this chapter as a supplement to general information contained in the *Getting Started with the Oracle Standard Management Pack*.

This chapter covers the following topics:

- Oracle Performance Manager Overview
- Starting Oracle Performance Manager
- Displaying Charts

See Also:

- *Oracle9i Real Application Clusters Installation and Configuration* for further information about installing Oracle Performance Manager
- *Oracle9i Database Performance Guide and Reference* for detailed information about the statistics these charts display and how to interpret these statistics
- *Oracle9i Database Reference* for more information about the fields in these charts and the V$ views from which they are derived
- *Oracle Enterprise Manager Administrator's Guide* for charts and statistics that are specific to **Oracle Enterprise Manager**

Oracle Performance Manager Overview

Oracle stores tuning and performance information for a Real Application Clusters database in a set of dynamic performance tables known as *V$ fixed views*. Each active **instance** has its own set of fixed views. You can use Oracle Performance Manager to query a global dynamic performance (GV$) view to retrieve the related V$ view information from all instances.

Oracle Performance Manager displays the retrieved information in a variety of tabular and graphic performance charts for Real Application Clusters. The statistics represent the aggregate performance of all instances of a **cluster** database running on cluster. The statistics are displayed in individual charts and include information about data block **ping**s, lock activity, file I/O, and session and user information. You can also use the Performance Manager to display an overview of several key statistics on one chart.

Performance monitoring is crucial for realizing the full potential of the system. There are several key performance metrics that you should constantly monitor to keep Real Application Clusters in peak operating condition. The Oracle Performance Manager, part of the Diagnostics Pack, is an available option to Oracle Enterprise Manager. It is an application designed to capture, compute, and present performance data that help database administrators analyze key performance metrics.

Oracle Performance Manager can be run with or without Oracle Enterprise Manager. If you choose to run it as a standalone product, Oracle Enterprise Manager does not have to be configured.

Real Application Clusters performance metrics are compiled into charts that are viewable with Oracle Performance Manager, as shown in Table 7–1.

Table 7–1 Performance Charts

Chart	Description
Total Ping Chart	Displays the ping count on all instances of the cluster database
Global Cache Timeouts Chart	Displays three time out and wait statistics
Global Cache CR Request Chart	Shows the average global cache consistent read (CR) request time, the **global cache cr timeouts**, and the **Global Cache Service Processes (LMSn)** use
Global Cache Lock Convert Chart	Shows the global cache lock converts for a cluster database
Instance Ping Chart	Identifies the instance that is contributing the most to the ping count
Global Cache CR Timeouts by Instance Chart	Displays global cache consistent read (CR) timeouts for all instances of a cluster database
Global Cache Freelist Waits by Instance Chart	Measures **global cache freelist waits** for all instances of a cluster database
Global Cache CR Request by Instance Chart	Displays global cache consistent read (CR) requests for all instances of a cluster database
Global Cache Lock Convert by Instance Chart	Displays row cache lock converts for all instances of a cluster database
Ping by File Chart	Identifies Ping count for all database files of the cluster database
File Ping by Instance Chart	Shows the total number of pings for a given file from each instance of a cluster database
Ping by Block Class Chart	Shows the ping count for all block classes of a cluster database
Ping by Object Chart	Shows the ping count for all database objects in a cluster database

Chart	Description
Object Ping by Instance Chart	Shows the ping count for a given database objects by each instance of a cluster database
Maximum Ping by Block Chart	Displays the most frequently pinged block numbers
Library Cache Lock Chart	Displays the waiting time for library cache lock for the entire cluster database
Library Cache Lock by Instance Chart	Shows library cache locks for all instances of a cluster database
Row Cache Lock Chart	This chart displays row cache locks for a cluster database (Data dictionary contention)
Row Cache Lock by Instance Chart	Displays row cache locks for all instances of a cluster database
Global Cache Current Block Request Chart	Displays the average global cache current block request time and average global cache current block serve time
Global Cache Current Block Request by Instance Chart	Displays the same information as the Global Cache Current Block Request Chart, but on a per instance basis
Global Cache Current Block Instance Activity Chart	Shows global cache current block requests for all instances of a cluster database. (Identifies the instance that is causing the maximum pinging activity through the **interconnect**)
File I/O Rate Default Chart	Displays the rate of **physical reads** and writes for all files in the cluster database
File I/O Rate by Object Default Chart	Displays the rate of reads and writes per datafile in the cluster database
File I/O Rate by Instance Default Chart	Displays the rate of reads and writes per instance in the cluster database
Lock Activity Default Chart	Displays the statistics on the lock activity rate for all the different lock types across all instances of the cluster database
Sessions Default Chart	Displays the sessions attached to the cluster database and related information
Users Default Chart	Displays the total number of user sessions logged on to the cluster database
Users Per Instance Default Chart	Displays the number of users logged on to the cluster for each instance

Starting Oracle Performance Manager

To use the Oracle Enterprise Manager **Console**, start the following components:

- Oracle Intelligent Agent
- Oracle Performance Manager

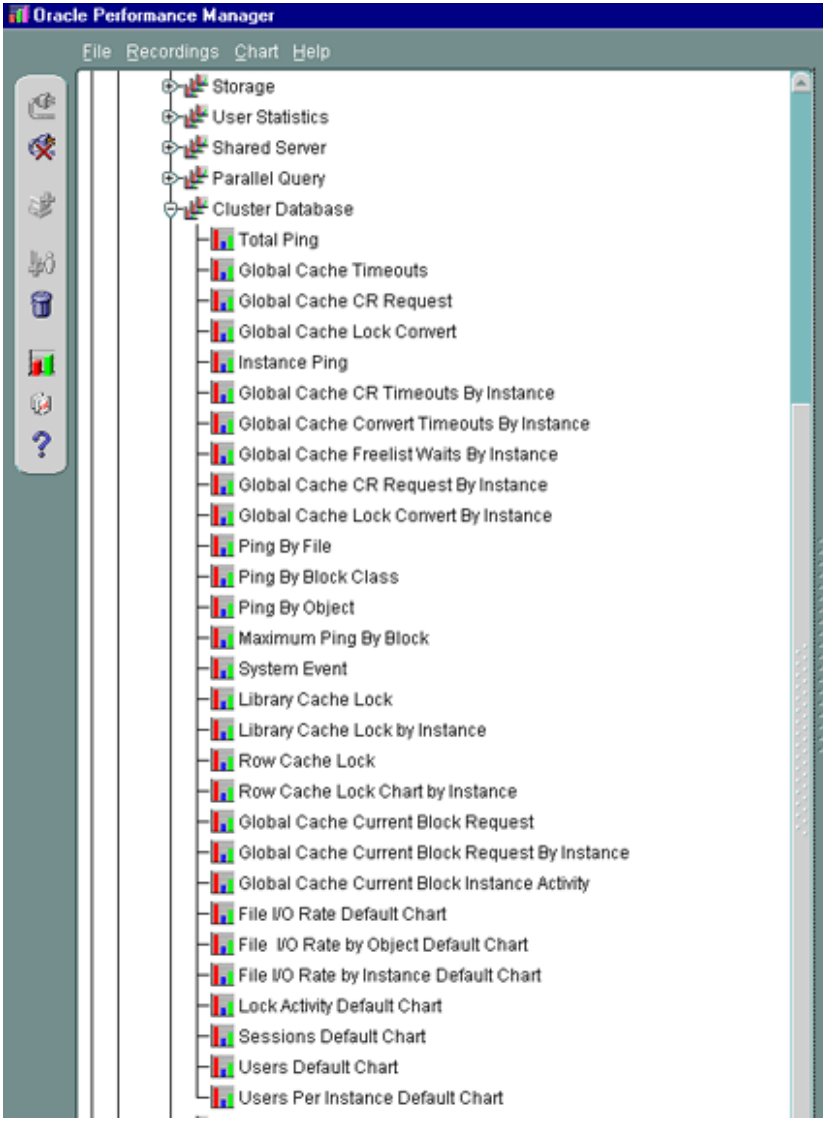
> **See Also:** *Oracle9i Real Application Clusters Installation and Configuration* for instructions

Displaying Charts

To display charts:

1. Follow the basic navigation procedures described in *Oracle9i Real Application Clusters Installation and Configuration*.
2. In the navigator, expand Databases or Clusters Instance > Cluster Database. You will see the list of available charts, as shown in Figure 7–1.

Figure 7–1 Expanding Charts



3. In the Oracle Clusters object folder, select a chart, then click Show Chart. The chart is displayed in a separate window.

Using the Statistics Charts

The following describes the use of the charts available in the current release. Note that if you are running software version 8.1.7 or earlier, you will have access to a different subset of these charts:

Total Ping Chart

This chart is used to get the forced disk write or *ping* count for a cluster database. When the ping count is converted to ping rate, an event is defined against the ping rate with a specified threshold limit. When ping rate exceeds this limit, the event will be triggered to notify you about excessive forced disk writing activity in the system.

Global Cache Timeouts Chart

This chart displays three time out and wait statistics on:

- Global cache CR timeouts
- Global cache convert timeouts
- Global cache freelist waits

These three statistics can be used to identify interconnect network congestion. See the descriptions for the Global Cache CR Timeouts by Instance, Global Cache Convert Timeouts by Instance, and Global Cache FreeList Waits by Instance charts for more information on these statistics.

Global Cache CR Request Chart

This chart shows, in milliseconds, for the entire cluster database:

- Average global cache CR request time
- Global cache CR timeouts
- Global Cache Service LMS process use

The average CR request time is defined as the ratio of sum of global cache CR block receive time and multiplied by 10 to the sum of **global cache cr blocks received**. The global cache CR block receive time is defined as total time taken for a **consistent read** request to complete. The global cache CR blocks received is defined as the count of CR blocks received from another instance when it cannot satisfy the CR block request from its local cache. The global cache CR timeouts refer to CR requests that have long delays and have timed out. The **Global Cache Service (GCS)** use is defined as the ratio of global cache CR block serve time to the **global**

cache current blocks served. These statistics are defined as follows: The global cache CR block serve time represents the accumulated time that it took the Global Cache Service process to serve a consistent read block request. For each request, the start time is recorded immediately after the Global Cache Service process takes a request off the request queue. The interval is completed after the block is sent. The **global cache cr blocks served** is the number of requests for a consistent read block served by the Global Cache Service process.

Global Cache Lock Convert Chart

This chart displays the average **global cache convert time** and average **global cache get time**, in milliseconds, across the entire cluster database. The **global cache gets** is defined as the count of new locks opened by an instance. The global cache get time is defined as the total amount of time including any waiting time taken to process a global cache get request. The **global cache converts** is defined as the count of lock conversions for existing locks for an instance. The global cache convert time is defined as the total amount of time including any waiting time taken to process a global cache convert request. The average convert time is defined as the ratio of sum of global cache convert time and multiplied by 10 to the sum of global cache converts for all instances of a cluster database. The average get time is defined as the ratio of sum of global cache get time and multiplied by 10 to the sum of global cache gets for all instances of a cluster database.

Instance Ping Chart

This chart is a drilldown from the Total Ping Chart. It is used to identify the instance that is contributing the most to the ping count. If certain instances generate more pings than others, this can indicate that there is contention among their workloads.

Global Cache CR Timeouts by Instance Chart

This chart measures the number of requests for a consistent read (CR) block that have long delays and have timed out. If the rate of change of global cache CR timeouts is more than zero for any instance, then this indicates that it is experiencing high IPC, slow interconnect network, or dropped network packets. When this happens, an EM event related to IPC issues will be generated.

Global Cache Convert Timeouts by Instance Chart

This chart displays the number of times a lock convert request by an instance in the Global Cache Service is timed out. This is useful for measuring high contention or I/O workload on the disk. If the rate of change of **global cache convert timeouts** is more than zero for any instance, then this indicates that it is experiencing high I/O

workload activity on the disk. When this happens you will be alerted to look into disk problems, deadlocks, and so forth.

Global Cache Freelist Waits by Instance Chart

This chart measures the usage of releasable locks. If the rate of change of global cache freelist waits is more than zero for any instance, then this indicates that more locks are consumed than are released by LMS process.

Global Cache CR Request by Instance Chart

This chart displays the same information as in Global Cache CR Request Chart, on per instance basis. The only difference between this chart and the Global Cache CR Request Chart is that these statistics should be read *per instance* and not for the entire cluster database.

Global Cache Lock Convert by Instance Chart

This chart displays the same information as in Global Cache Lock Convert chart, on a per instance basis. Here the average convert time is defined as the ratio of sum of global cache convert time and multiplied by 10 to the sum of global cache converts for each instance of a cluster database. Similarly, the average get time is defined as the ratio of sum of global cache get time and multiplied by 10 to the sum of global cache gets for each instance of a cluster database.

Ping by File Chart

This chart displays the total number of pings for all **datafiles** of the cluster database. By using this chart you can identify the database file that has the maximum ping count. You should consider physically distributing these files to reduce I/O delays. Another option would be to partition the application to increase the percentage of local work on the objects contained in the files to reduce the excessive pinging.

File Ping by Instance Chart

This chart is a drilldown from the Ping by File chart. Having identified the file with highest ping count, by using this chart you can determine how much each instance is contributing to this file's ping count. Once the instance has been identified, you should consider localizing the contents of this file to the **node** on which this instance runs for reducing the pings.

Ping by Block Class Chart

This chart displays the total number of pings experienced by each block class for all instances of a cluster database. Once the top pinged block class has been identified, use the V$PING view to obtain details about a particular table or index and the file and block numbers on which there is significant lock conversion activity. An additional drilldown chart (not visible at the top level) is available. It displays the total number of pings experienced by each block class for a given instance of a cluster database.

Ping by Object Chart

This chart is displayed as a drill-down from Instance Ping chart. This chart displays the number of pings for all database objects in a given file by each instance of a cluster database. The database objects with the most pings are considered *hot spots* and sources of performance problems. You should consider identifying the transactions that are accessing these hot spot objects by querying V$SQLAREA view to get the sql_text. After the transactions are identified, you should consider binding these transactions to a single instance to reduce cross-pinging from other instances. Another option would be to partition the data in this object by using hashing so that each transaction gets its data from its partitioned data range.

Object Ping by Instance Chart

This chart is a drilldown from Ping by Object chart. This chart displays the total number of pings contributed by each instance of a cluster database to the given database object. This helps in identifying the instance that is causing the most pings for a given database object. Once the instance has been identified, you should consider making this object data locally available to the instance.

Maximum Ping by Block Chart

This chart displays the most frequently pinged block numbers for a given database object in a database file. The most pinged block numbers are the hot spots and sources of contention between instances. After identifying these block numbers, you should consider reducing the number of rows contained in these blocks and redistributing the data. One way to do this would be to reduce the database block size.

Library Cache Lock Chart

This chart displays the waiting time for library cache objects. This wait time can be high due to parsing or invalidating cursors. Library Cache Lock displays DLM_LOCK_REQUESTS, DLM_PIN_REQUESTS, and DLM_INVALIDATIONS statistics for

the entire cluster database. These statistics are defined as follows: The DLM_LOCK_REQUESTS statistics represents the number of times a lock was requested for a database object. The DLM_PIN_REQUESTS statistics represents the number of times a pin was requested for a database object. The DLM_INVALIDATIONS statistic represents the number of invalidation pings received from other instances.

Library Cache Lock by Instance Chart

This chart displays the same information as the Library Cache Lock chart, on per instance basis. The only important difference in the definition is that here these statistics should be read as per instance and not for the entire cluster database.

Row Cache Lock Chart

This chart displays DLM_REQUESTS, DLM_CONFLICTS, and percentage of DLM_REQUESTS that result in DLM_CONFLICTS for the whole cluster database. These are useful for analyzing dictionary contention. The DLM_REQUESTS represents the number of DLM requests. A DLM request for a lock is issued for each consistent-read block request. The DLM_CONFLICTS represents the number of DLM lock request conflicts. A DLM conflict occurs when another instance already owned a lock on a CR block in mode that conflicts with the mode requested by this instance. The percentage value is computed as the ratio of total number of DLM_CONFLICTS to the total number of DLM_REQUESTS for all instances of a cluster database.

Row Cache Lock by Instance Chart

This chart displays the same information as Row Cache Lock Chart, but on a per instance basis. The only important difference in the definition is that here these statistics should be read as per instance and not for the entire cluster database.

Global Cache Current Block Request Chart

This chart displays the average global cache current block request time and average global cache current block serve time, in milliseconds, for the entire cluster database. The average global cache current block request time is defined as the ratio of **global cache current block receive time** and multiplied by 10 to the **global cache current blocks received**. The average global cache current block serve time is defined as the ratio of global cache current block serve time and multiplied by 10 to the global cache current blocks served.

Global Cache Current Block Request by Instance Chart

This chart displays the same information as Global Cache Current Block Request Chart, but on a per instance basis. The only important difference in the definition is that here these statistics should be read as per instance and not for the entire cluster database.

Global Cache Current Block Instance Activity Chart

This chart can be used to determine how this instance's pings are themselves distributed across the interconnect. This chart displays **global cache current block pin time**, **global cache current block flush time**, and **global cache current block send time**, in milliseconds, for the given instance. This helps in pin pointing the cause of excessive pinging activity for the given instance, such as whether the given instance is spending too much time in block pinning to the local cache, block flushing to the disk, or block sending across the interconnect.

File I/O Rate Default Chart

This chart displays the rate of physical reads and writes for all files in the database. You can drill down to obtain the same information either at the instance level or at the file level

File I/O Rate by Object Default Chart

This chart displays the rate of reads and writes per datafile in the database

File I/O Rate by Instance Default Chart

This chart displays the rate of reads and writes per instance in the database

Lock Activity Default Chart

This chart displays the statistics on the lock activity rate for all the different lock types across the cluster. You can drill down to obtain lock activity information for a particular lock type at the instance level. The global cache usability enhancements in Oracle9*i* Release 1 (9.0.1) simplified the lock activity histograms.

Sessions Default Chart

This chart displays the sessions attached to the cluster database and related information, such as **instance name**, session ID, session serial number, process ID, status, and user name

Users Default Chart

This chart displays the total number of user sessions logged on to the cluster database, regardless of whether activity is generated. This information is also available for each instance.

Users Per Instance Default Chart

This chart displays the number of users logged on to each instance

Active Users Chart

This chart displays the total number of active users on the cluster database

Active Users by Instance Chart

This chart displays the number of active users on each instance of the cluster database.

Clusters Data Block Ping by Instance Chart

This chart displays the block pings per instance in the cluster database

Part V

Real Application Clusters Reference

Part Five contains **Oracle Real Application Clusters** reference information. The Appendices in Part Five are:

- Appendix A, "Configuring Multi-Block Lock Assignments (Optional)"
- Appendix B, "A Case Study in Real Application Clusters Database Design"
- Glossary

A

Configuring Multi-Block Lock Assignments (Optional)

This appendix explains how to configure locks to cover multiple blocks. Refer to this appendix only for rare circumstances to override **Oracle Real Application Clusters**' default resource control scheme as performed by the **Global Cache Service (GCS)** and the **Global Enqueue Service (GES)**. The topics in this appendix are:

- Before You Override the Global Cache and Global Enqueue Service Resource Control Mechanisms
- Deciding Whether to Override Global Cache Service and Global Enqueue Service Processing
- Setting GC_FILES_TO_LOCKS
- Additional Considerations for Setting GC_FILES_TO_LOCKS
- Database Design Considerations and Free List Groups
- Tuning Parallel Execution on Real Application Clusters
- Analyzing Real Application Clusters I/O Statistics
- Monitoring Multi-Block Lock Usage by Detecting False Forced Writes
- Lock Names and Lock Formats

Before You Override the Global Cache and Global Enqueue Service Resource Control Mechanisms

Oracle strongly recommends that you avoid overriding the resource control performed by the Global Cache and Global Enqueue Services. The default scheme provides exceptional performance for almost all system types in almost all Real Application Clusters environments. In addition, assigning locks requires additional administrative and tuning effort. Therefore, using the default scheme is preferable to performing the complex tasks required to override the default strategy as described in this appendix.

Note: Only use the information in this appendix for exceptional cases. An example of this is an application where the data access patterns are almost exclusively read-mostly.

Deciding Whether to Override Global Cache Service and Global Enqueue Service Processing

Cache Fusion provides exceptional scalability and performance using cache-to-cache transfers of data that is not cached locally. In other words, before an **instance** reads a data block from disk, Oracle attempts to obtain the requested data from another instance's cache. If the requested block exists in another cache, then the data block is transferred across the **interconnect** from the holding instance to the requesting instance.

Real Application Clusters' resource control scheme guarantees the integrity of changes to data made by multiple instances. By default, each data block in an instance's buffer cache is protected by the Global Cache Service. The GCS tracks the access modes, roles, privileges, and states of these resources.

In rare situations, you may want to override the GCS, and the Global Enqueue Service by configuring multi-block locks where one lock covers multiple data blocks in a file. If blocks are frequently accessed from the same instance, or if blocks are accessed from multiple **node**s but in compatible modes such as *shared* mode for concurrent reads, then a lock configuration may improve performance.

To do this, set the `GC_FILES_TO_LOCKS` parameter and specify the number of locks that Oracle uses for particular files. The syntax of the parameter also enables you to specify lock allocations for groups of files as well as the number of contiguous data blocks to be covered by each lock. If you indiscriminately use

values for GC_FILES_TO_LOCKS, then adverse performance such as excessive **forced disk writes** can result. Therefore, only set GC_FILES_TO_LOCKS for:

- Read-only or read-mostly files and **tablespaces**
- Partitioned data access, where data in a particular file are only or mostly modified by one instance
- Datafiles other than those associated with temporary tablespaces, tablespaces marked as READ ONLY, and tablespace containing **rollback segments**

When to Use Locks

Using multiple locks for each file can be useful for the types of data shown in Table A–1.

Table A–1 When to Use Locks

Situation	Reason
When the data is mostly read-only.	A few locks can cover many blocks without requiring frequent lock operations. These locks are released only when another instance needs to modify the data. Assigning locks can result in better performance on read-only data with **parallel execution** processing. If the data is strictly read-only, then consider designating the tablespace as read-only.
When the data can be partitioned according to the instance which is likely to modify it.	Lock assignments that you define to match this partitioning scheme allow instances to hold disjoint sets of locks. This reduces the need for lock operations.
When a large amount of data is modified by a relatively small set of instances.	Lock assignments permit access to an un-cached database block to proceed without Parallel Cache Management activity. However, this is only possible if the block is already in the requesting instance's cache.

Using locking can cause additional cross-instance cache management activity because conflicts can occur between instances that modify different database blocks. Resolution of false forced disk writes or excessive forced disk writes can require writing several blocks from the cache of the instance that currently owns access to the blocks.

Setting GC_FILES_TO_LOCKS

Set the GC_FILES_TO_LOCKS initialization parameter to specify the number of locks covering data blocks in a datafile or set of datafiles. This section covers:

- GC_FILES_TO_LOCKS Syntax
- Lock Assignment Examples
- Blocking Factor, Extent Allocation, and Free List Groups

GC_FILES_TO_LOCKS Syntax

The syntax for the GC_FILES_TO_LOCKS parameter enables you to specify the relationship between locks and files. The syntax for this parameter is:

GC_FILES_TO_LOCKS="{*file_list*=*#locks*[!*blocks*] [EACH][:]} . . ."

Where:

file_list	*file_list* specifies a single file, range of files, or list of files and ranges as follows: *fileidA*[-*fileidC*][,*fileidE*[-*fileidG*]] ... Query the data dictionary view DBA_DATA_FILES to find the correspondence between file names and file ID numbers.
#locks	Sets the number of locks to assign to file_list.
!*blocks*	Specifies the number of contiguous data blocks to be covered by each lock; also called the *blocking factor*
EACH	Specifies *#locks* as the number of locks to be allocated to *each* file in file_list.

Note: All instance must have identical values for GC_FILE_TO_LOCKS. Also, do not use spaces within the quotation marks of the GC_FILES_TO_LOCKS parameter syntax.

The default value for *!blocks* is 1. When you specify *blocks*, contiguous data blocks are covered by each of the *#lock* locks. To specify a value for *blocks*, use the exclamation point (!) separator. You would primarily specify *blocks*, and not specify the EACH keyword to allocate sets of locks to cover multiple datafiles.

Always set the *!blocks* value to avoid interfering with the data partitioning gained by using free list groups. Normally you do not need to preallocate extents. When a

row is inserted into a table and Oracle allocates new extents, Oracle allocates contiguous blocks that are specified with *!blocks* in GC_FILES_TO_LOCKS to the free list group associated with an instance.

Lock Assignment Examples

For example, you can assign 300 locks to file 1 and 100 locks to file 2 by adding the following line to your **initialization parameter file**:

```
GC_FILES_TO_LOCKS = "1=300:2=100"
```

The following entry specifies a total of 1500 locks: 500 each for files 1, 2, and 3:

```
GC_FILES_TO_LOCKS = "1-3=500EACH"
```

By contrast, the following entry specifies a total of only 500 locks spread across the three files:

```
GC_FILES_TO_LOCKS = "1-3=500"
```

The following entry indicates that Oracle should use 1000 distinct locks to protect file 1. The data in the files is protected in groups of 25 contiguous locks.

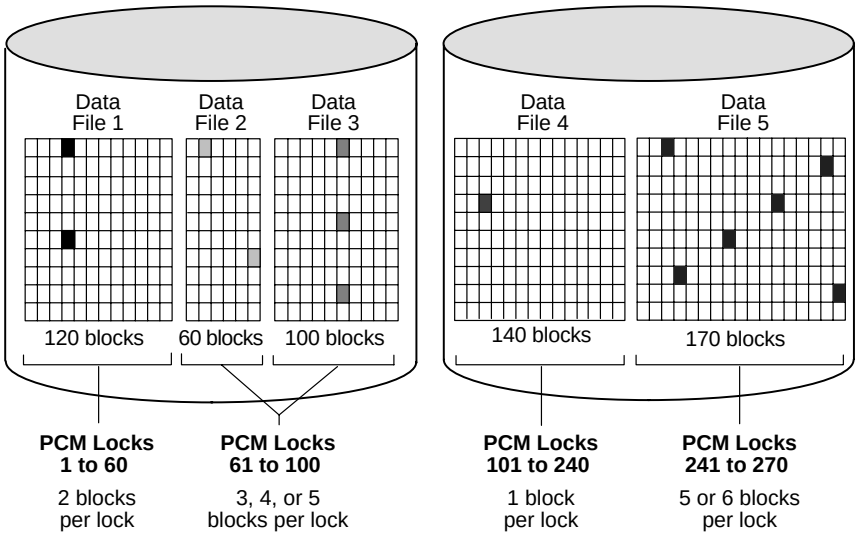
```
GC_FILES_TO_LOCKS = "1=1000!25"
```

If you define a datafile with the AUTOEXTEND clause or if you issue the ALTER DATABASE ... DATAFILE ... RESIZE statement, then you may also need to adjust the lock assignment.

When you add new datafiles, decide whether these new files should be subject to the default control of the GCS or whether you want to assign locks using the GC_FILES_TO_LOCKS initialization parameter.

The following examples show different methods of mapping blocks to locks and how the same locks are used on multiple datafiles.

Figure A–1 Mapping Locks to Data Blocks



Example 1 Figure A–1 shows an example of mapping blocks to locks for the parameter value `GC_FILES_TO_LOCKS = "1=60:2-3=40:4=140:5=30"`.

In datafile 1 shown in Figure A–1, 60 locks map to 120 blocks, which is a multiple of 60. Each lock covers two data blocks.

In datafiles 2 and 3, 40 locks map to a total of 160 blocks. A lock can cover either one or two data blocks in datafile 2, and two or three data blocks in datafile 3. Thus, one lock can cover three, four, or five data blocks across both datafiles.

In datafile 4, each lock maps exactly to a single data block, since there is the same number of locks as data blocks.

In datafile 5, 30 locks map to 170 blocks, which is not a multiple of 30. Each lock therefore covers five or six data blocks.

Each lock illustrated in Figure A–1 can be held in either shared read mode or read-exclusive mode.

Example 2 The following parameter setting allocates 500 locks to datafile 1; 400 locks each to files 2, 3, 4, 10, 11, and 12; 150 locks to file 5; 250 locks to file 6; and 300 locks collectively to files 7 through 9:

```
GC_FILES_TO_LOCKS = "1=500:2-4,10-12=400EACH:5=150:6=250:7-9=300"
```

This example assigns a total of (500 + (6*400) + 150 + 250 + 300) = 3600 locks. You can specify more than this number of locks if you add more datafiles.

Example 3 In Example 2, 300 locks are allocated to datafiles 7, 8, and 9 collectively with the clause "7-9=300". The keyword EACH is omitted. If each of these datafiles contains 900 data blocks, then for a total of 2700 data blocks, then each lock covers nine data blocks. Because the datafiles are multiples of 300, the nine locks cover three data blocks in each datafile.

Example 4 The following parameter value allocates 200 locks each to files 1 through 3; 50 locks to datafile 4; 100 locks collectively to datafiles 5, 6, 7, and 9; and 20 locks in contiguous 50-block groups to datafiles 8 and 10 combined:

```
GC_FILES_TO_LOCKS = "1-3=200EACH 4=50:5-7,9=100:8,10=20!50"
```

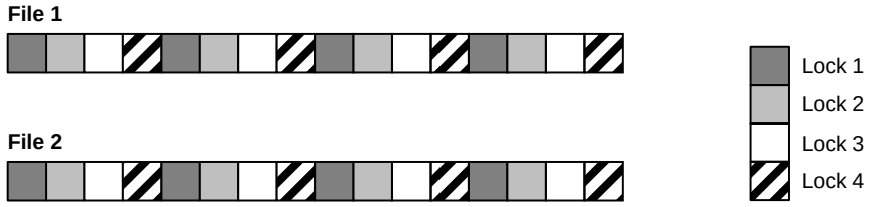
In this example, a lock assigned to the combined datafiles 5, 6, 7, and 9 covers one or more data blocks in each datafile, unless a datafile contains fewer than 100 data blocks. If datafiles 5 to 7 contain 500 data blocks each and datafile 9 contains 100 data blocks, then each lock covers 16 data blocks: one in datafile 9 and five each in the other datafiles. Alternatively, if datafile 9 contained 50 data blocks, half of the locks would cover 16 data blocks (one in datafile 9); the other half of the locks would only cover 15 data blocks (none in datafile 9).

The 20 locks assigned collectively to datafiles 8 and 10 cover contiguous groups of 50 data blocks. If the datafiles contain multiples of 50 data blocks and the total number of data blocks is not greater than 20 times 50, that is, 1000, then each lock covers data blocks in either datafile 8 or datafile 10, but not in both. This is because each of these locks covers 50 contiguous data blocks. If the size of datafile 8 is not a multiple of 50 data blocks, then one lock must cover data blocks in both files. If the sizes of datafiles 8 and 10 exceed 1000 data blocks, then some locks must cover more than one group of 50 data blocks, and the groups might be in different files.

Example 5 GC_FILES_TO_LOCKS="1-2=4"

In this example, four locks are specified for files 1 and 2. Therefore, the number of blocks covered by each lock is eight ((16+16) / 4). The blocks are not contiguous.

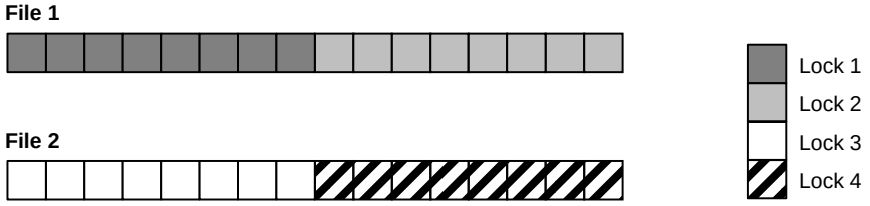
Figure A–2 GC_FILES_TO_LOCKS Example 5



Example 6 GC_FILES_TO_LOCKS="1-2=4!8"

In this example, four locks are specified for files 1 and 2. However, the locks must cover eight contiguous blocks.

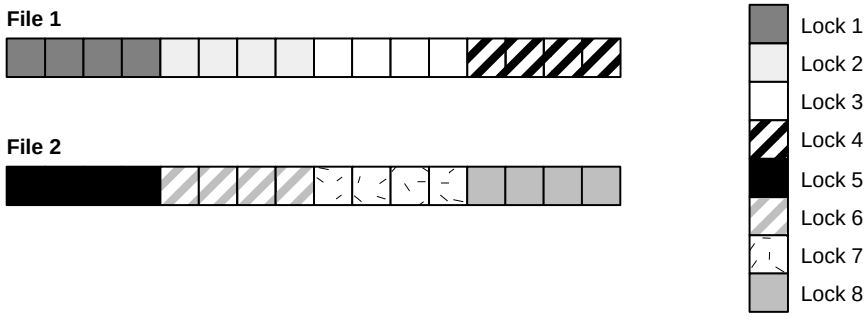
Figure A–3 GC_FILES_TO_LOCKS Example 6



Example 7 GC_FILES_TO_LOCKS="1-2=4!4EACH"

In this example, four locks are specified for file 1 and four for file 2. The locks must cover four contiguous blocks.

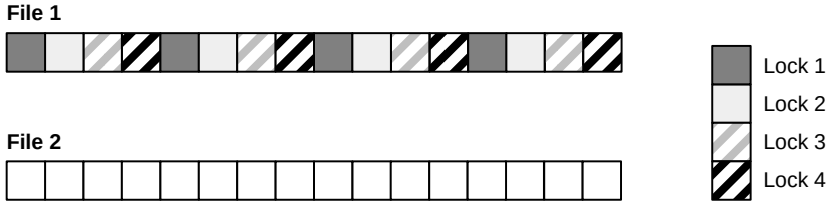
Figure A–4 GC_FILES_TO_LOCKS Example 7



Example 8 GC_FILES_TO_LOCKS="1=4:2=0"

In this example, file 1 has multi-block lock control with 4 locks. On file 2, locks are allocated.

Figure A–5 GC_FILES_TO_LOCKS Example 8



Blocking Factor, Extent Allocation, and Free List Groups

Use the !*blocks* option of GC_FILES_TO_LOCKS to align the extents of contiguous blocks allocated to an object with lock coverage. When using the *!blocks* notation, contiguous data blocks are covered by one lock. For example:

```
GC_FILES_TO_LOCKS="12=1000!25"
```

Allocates 1000 locks to file 12 with a periodicity of 25, in other words one lock covers 25 contiguous blocks.

If an extent definition (INITIAL EXTENT, NEXT EXTENT) for a table corresponds to !blocks, then in this case the 25 contiguous blocks covered by a lock coincide with the extent boundaries. In other words, all the blocks covered by a multi-block lock are in the same extent.

This is important when using free list groups. When no more blocks exist with free space, Oracle allocates a new extent. The blocks in this extent are then allocated to a particular free list group. If not properly configured, then locks can also cover blocks from another extent which might be in the free list group used by another instance. This results in *false* forced disk writes.

If you do not use the blocking factor as described in this section, then the same lock can cover blocks from different extents allocated to distinct free list groups, thus incurring additional overhead. This situation is what free list groups are supposed to avoid.

Dynamic Allocation of Blocks on Lock Boundaries

To accommodate growth, the strategy of dynamically allocating blocks to free list groups is more effective than the preallocation of extents.

> **See Also:** "Extent Preallocation Examples" on page 4-11 before using the methods described in this section

You can also use the !*blocks* option of GC_FILES_TO_LOCKS to dynamically allocate blocks to a free list from the **high water mark** within a lock boundary. This method does not eliminate *all* forced writes on the segment header. Instead, this method allocates blocks as needed so you do not have to preallocate extents.

Because locks are owned by instances, blocks are allocated on a per-instance basis—and that is why they are allocated to free list groups. Within an instance, blocks can be allocated to different free lists.

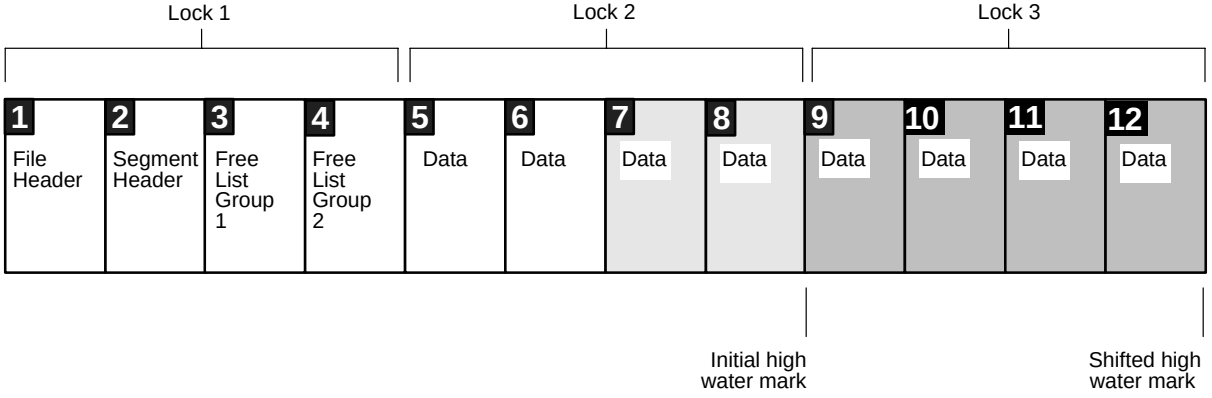
Using this method, you can either explicitly allocate the !*blocks* value, or leave the balance of new blocks covered by the existing lock. If you choose the latter, there still may be contention for the existing locks by allocation to other instances.

Moving a Segment's High Water Mark

A segment's high water mark is the current limit to the number of blocks that have been allocated within the segment. If you are allocating extents dynamically, the high water mark is also the lock boundary. The lock boundary and the number of blocks that will be allocated at one time within an extent must coincide. This value must be the same for all instances.

Consider the example in Figure A–6 with four blocks for each lock (!4). Locks have been allocated before the block content has been entered. If Oracle fills data block D2, held by Lock 2, and then allocated another range of four blocks, only the number of blocks fitting within the lock boundary are allocated. In this case, this includes blocks 7 and 8. Both of these are protected by the current lock. With the high water mark at 8, when instance 2 allocates a range of blocks, all four blocks 9 to 12 are allocated, covered by Lock 3. The next time instance 1 allocates blocks it will get blocks 13 to 16, covered by Lock 4.

Figure A–6 A File with a High Water Mark That Moves as Oracle Allocates Blocks



Example This example assumes that `GC_FILES_TO_LOCKS` has the following setting for both instances:

```
GC_FILES_TO_LOCKS = "1000!5"
```

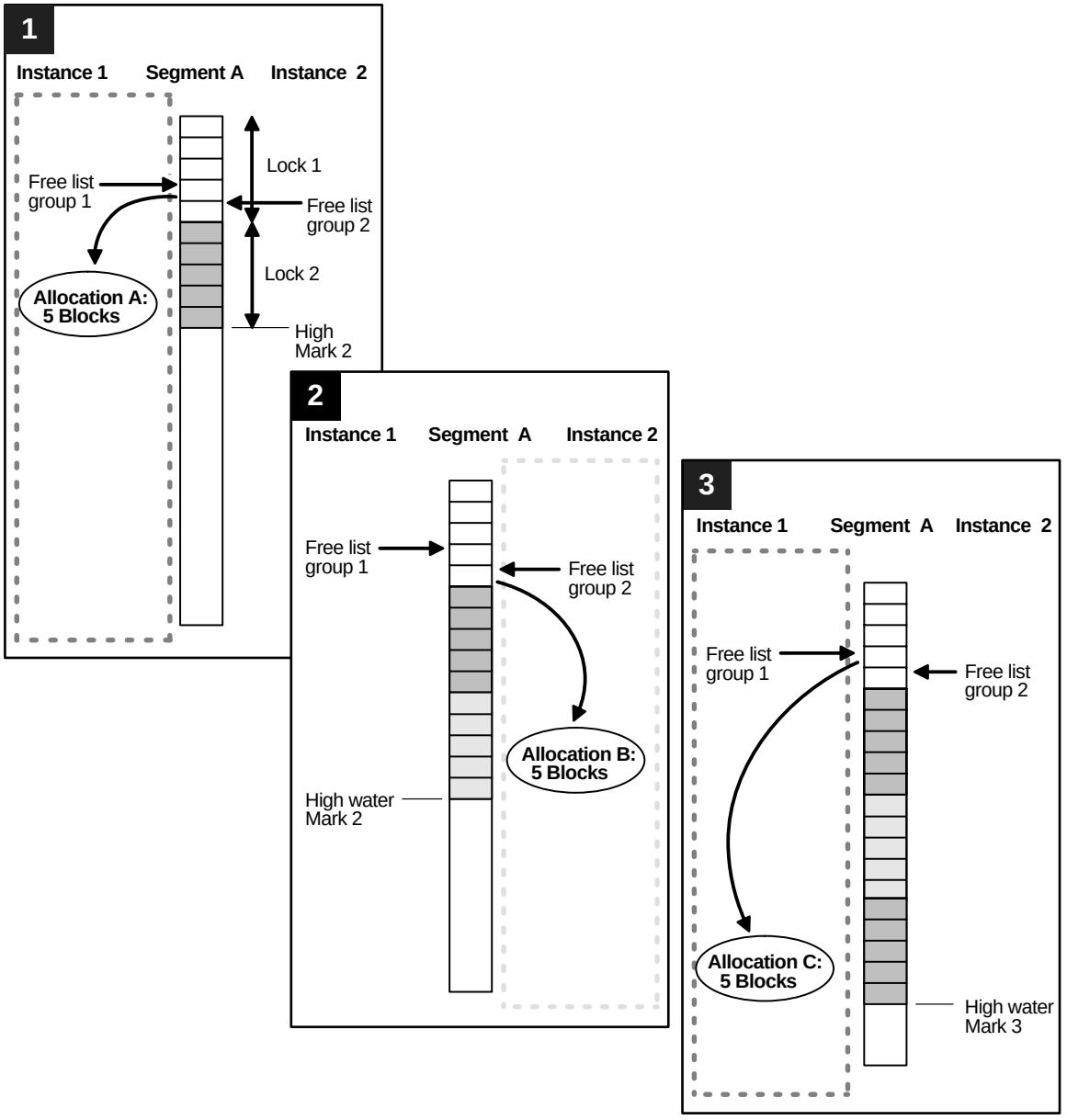
With the EACH option specified, each file in `file_list` is allocated `#locks` number of locks. Within each file, `!blocks` specifies the number of contiguous data blocks to be covered by each lock.

Figure A–7 shows the incremental process by which the segment grows:

- Stage 1 shows an extent in which instance 1 allocates 5 data blocks, which are protected by Lock 2.
- Stage 2 shows instance 2 allocating 5 more data blocks, protected by Lock 3.
- Stage 3 shows instance 1 once more allocating 5 data blocks, protected by Lock 4.

In this way, if user A on Instance 1 is working on block 10, no one else from either instance can work on any block in the range of blocks covered by Lock 2. This includes blocks 6 through 10.

Figure A–7 Allocating Blocks within An Extent



Additional Considerations for Setting GC_FILES_TO_LOCKS

Setting GC_FILES_TO_LOCKS in Real Application Clusters has further implications. For example, setting it can increase monitoring overhead and you may have to frequently adjust the parameter when the database grows or when you add files. Moreover, you cannot dynamically change the setting for GC_FILES_TO_LOCKS. To change the setting, you must stop the instances, alter the setting, and restart all the instances. In addition, consider the following topics in this section:

- Expanding or Adding Datafiles
- Files To Avoid Including in GC_FILES_TO_LOCKS Settings

Expanding or Adding Datafiles

Sites that run continuously cannot afford to shut down for parameter value adjustments. Therefore, when you use the GC_FILES_TO_LOCKS parameter, remember to provide room for growth or room for files to extend.

You must also carefully consider how you use locks on files that do not grow significantly, such as read-only or read-mostly files. It is possible that better performance would result from assigning fewer locks for multiple blocks. However, if the expected CPU and memory savings due to fewer locks do not outweigh the administrative overhead, use the resource control scheme of the Global Cache and Global Enqueue Services.

Files To Avoid Including in GC_FILES_TO_LOCKS Settings

Never include the following types of files in the GC_FILES_TO_LOCKS parameter list:

- Files that contain rollback segments.
- Files that are part of a TEMPORARY tablespace.
- Files with read-only data within a tablespace that is explicitly set to READ ONLY; the exception to this is a single lock that you can assign to ensure the tablespace does not have to contend for spare locks—but setting this lock is not mandatory—you can still leave this tablespace unassigned.

Database Design Considerations and Free List Groups

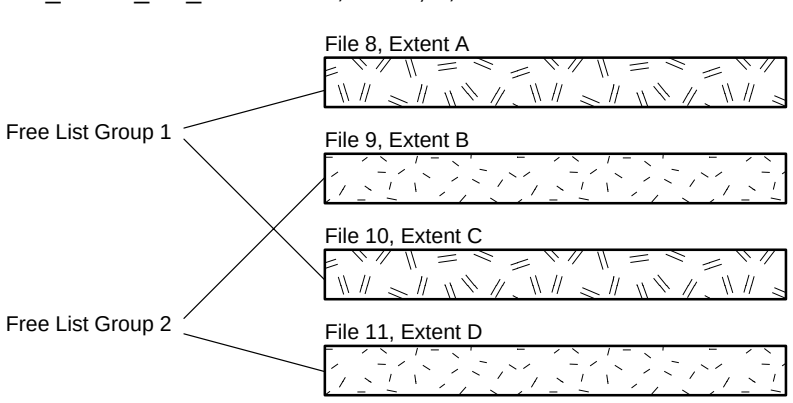
When data is frequently inserted into a table from multiple nodes and the table is not partitioned, you can use free list groups to avoid performance issues. Free list groups separate the data structures associated with the free space management of a table into disjoint sets that are available for individual instances. However, if you override the Global Cache and Global Enqueue Services, consider assigning locks with free lists as described under the following topic.

Associating Locks with Free Lists

If each extent in a table is in a separate datafile, you can use the GC_FILES_TO_LOCKS parameter to allocate specific ranges of locks to each extent, so that each set of locks is associated with only one group of free lists.

Figure A–8 shows multiple extents in separate files. The GC_FILES_TO_LOCKS parameter allocates 10 locks to files 8 and 10, and 10 locks to files 9 and 11. Extents A and C are in the same free list group, and extents B and D are in another free list group. One set of locks is associated with files 8 and 10, and a different set of locks is associated with files 9 and 11. You do not need separate locks for files that are in the same free list group, such as files 8 and 10, or files 9 and 11.

Figure A–8 Extents and Free List Groups



This example assumes total partitioning for reads as well as writes. If more than one instance updates blocks, then it is desirable to have more than one lock for each file to minimize forced reads and writes. This is because even with a shared lock, *all*

blocks held by a lock are subject to forced reads when another instance tries to read even *one* of the blocks held in exclusive mode.

Tuning Parallel Execution on Real Application Clusters

To optimize parallel execution in Real Application Clusters environments when not using the default resource control scheme, you must accurately set the `GC_FILES_TO_LOCKS` parameter. Data block address locking in its default behavior assigns one lock to each block. For example, during a full table scan, a lock must be acquired for each block read into the scan. To accelerate full table scans, you use one of the following three possibilities:

- For datafiles containing truly read-only data, set the tablespace to read only. Then lock operations do not occur.
- Alternatively, for data that is mostly read-only, assign very few hashed locks (for example, 2 shared locks) to each datafile. Then these are the only locks you have to acquire when you read the data.
- If you want data block address or fine-grain locking, group the blocks controlled by each lock, using the ! option. This has advantages over default data block address locking because with the default, you would need to acquire one million locks in order to read one million blocks. When you group the blocks, you reduce the number of locks allocated by the grouping factor. Thus a grouping of !10 would mean that you would only have to acquire one tenth as many locks as with the default. Performance improves due to the dramatically reduced amount of lock allocation. As a rule of thumb, performance with a grouping of !10 is comparable to the speed of hashed locking.

  To speed up parallel DML operations, consider using hashed locking or a high grouping factor rather than database address locking. A parallel execution server works on non-overlapping partitions; it is recommended that partitions not share files. You can thus reduce the number of lock operations by having only 1 hashed lock for each file. Because the parallel execution server only works on non-overlapping files, there are no lock pings.

The following guidelines affect memory usage, and thus indirectly affect performance:

- Never allocate locks for **datafiles** of temporary tablespaces.
- Allocate specific locks for the SYSTEM tablespace. This practice ensures that data dictionary activity such as space management never interferes with the data tablespaces at a cache management level (error 1575).

Analyzing Real Application Clusters I/O Statistics

If you set GC_FILES_TO_LOCKS, then Cache Fusion is disabled. In this case, you can use three statistics in the V$SYSSTAT view to measure the I/O workload related to global cache synchronization:

- DBWR cross-instance writes
- Remote instance undo header writes
- Remote instance undo block writes

DBWR cross-instance writes occur when Oracle resolves inter-instance data block contention by writing the requested block to disk before the requesting node can use it.

Cache Fusion eliminates the disk I/O for current and consistent-read versions of blocks. This can lead to a substantial reduction in **physical writes** and reads performed by each instance.

Analyzing Real Application Clusters I/O Statistics Using V$SYSSTAT

You can obtain the following statistics to quantify the write I/Os required for global cache synchronization.

1. Use this syntax to query the V$SYSSTAT view:

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ('DBWR cross-instance writes',
'remote instance undo block writes',
'remote instance undo header writes',
'physical writes');
```

Oracle responds with output similar to:

```
NAME                                                          VALUE
------------------------------------------------------------ ----------
physical writes                                               41802
DBWR cross-instance writes                                     5403
remote instance undo block writes                                 0
remote instance undo header writes                                2
4 rows selected.
```

Where the statistic *physical writes* refers to all physical writes that occurred from a particular instance performed by DBWR, the value for **DBWR cross-instance writes** accounts for all writes caused by writing a dirty buffer containing a data

block that is requested for modification by another instance. Because the DBWR process also handles cross-instance writes, *DBWR cross-instance writes* are a subset of all *physical writes*.

2. Calculate the ratio of Real Application Clusters-related I/O to overall physical I/O using this equation:

$$\frac{\textit{DBWR cross-instance writes}}{\textit{physical writes}}$$

3. Use this equation to calculate how many writes to rollback segments occur when a remote instance needs to read from rollback segments that are in use by a local instance:

$$\frac{\textit{(remote instance undo header writes + remote instance undo block writes)}}{\textit{DBWR cross-instance writes}}$$

The ratio shows how much disk I/O is related to writes to rollback segments.

4. To estimate the number or percentage of reads due to global cache synchronization, use the number of lock requests for conversions from NULL(N) to Shared mode (S) counted in `V$LOCK_ACTIVITY` and the **physical reads** statistics from `V$SYSSTAT`.

The following formula computes the percentage of reads that are only for local work where **lock buffers for read** represents the N-to-S block access mode conversions:

$$\frac{\textit{(physical reads - (lock buffers for read))} * 100}{\textit{physical reads}}$$

These so-called *forced reads* occur when a cached data block that was previously modified by the local instance had to be written to disk. This is due to a request from another instance, so the block is then re-acquired by the local instance for a read.

Monitoring Multi-Block Lock Usage by Detecting False Forced Writes

False forced writes occur when Oracle down-converts a lock that protects two or more blocks if the blocks are concurrently updated from different nodes. Assume that each node is updating a different block covered by the same lock. In this case, each node must write both blocks to disk even though the node is updating only one of them. This is necessary because the same lock covers both blocks.

Statistics are not available to show false forced write activity. To assess false forced write activity you can only consider circumstantial evidence as described in this section.

The following SQL statement shows the number of lock operations causing a write, and the number of blocks actually written:

```
SELECT VALUE/(A.COUNTER + B.COUNTER + C.COUNTER) "PING RATE"
  FROM V$SYSSTAT,
    V$LOCK_ACTIVITY A,
    V$LOCK_ACTIVITY B,
    V$LOCK_ACTIVITY C
WHERE A.FROM_VAL = 'X'
    AND A.TO_VAL = 'NULL'
    AND B.FROM_VAL = 'X'
    AND B.TO_VAL = 'S'
    AND C.FROM_VAL = 'X'
    AND C.TO_VAL = 'SSX'
    AND NAME = 'DBWR cross-instance writes';
```

Table A–2 shows how to interpret the forced disk write rate.

Table A–2 Interpreting the Forced Write Rate

Forced Disk Write Rate	Meaning
Less than 1	False forced writes may be occurring, but there are more lock operations than forced disk writes. DBWR is writing blocks fast enough, resulting in no writes for lock activity. This is also known as a *soft ping*, meaning I/O activity is not required for the forced disk write, only lock activity.
Equal to 1	Each lock activity involving a potential write causes the write to occur. False forced writes may be occurring.
Greater than 1	False forced writes are definitely occurring.

Use this formula to calculate the percentage of false forced writes:

$$\frac{(ping\_rate - 1)}{ping\_rate} * 100$$

Then check the total number of writes and calculate the number due to false forced writes:

```
SELECT Y.VALUE "ALL WRITES",
    Z.VALUE "PING WRITES",
    Z.VALUE * pingrate "FALSE PINGS",
FROM V$SYSSTAT Z,
    V$SYSSTAT Y,
WHERE Z.NAME = 'DBWR cross-instance writes'
AND Y.NAME = 'physical writes';
```

Here, *ping_rate* is given by the following SQL statement:

```
CREATE OR REPLACE VIEW PING_RATE AS
SELECT ((VALUE/(A.COUNTER+B.COUNTER+C.COUNTER))-1)/
    (VALUE/(A.COUNTER+B.COUNTER+C.COUNTER)) RATE
FROM V$SYSSTAT,
    V$LOCK_ACTIVITY A,
    V$LOCK_ACTIVITY B,
    V$LOCK_ACTIVITY C
WHERE A.FROM_VAL = 'X'
    AND A.TO_VAL   = 'NULL'
    AND B.FROM_VAL = 'X'
    AND B.TO_VAL   = 'S'
    AND C.FROM_VAL = 'X'
AND C.TO_VAL   = 'SSX'
AND NAME = 'DBWR cross-instance writes';
```

The goal is not only to reduce overall forced disk writes, but also to reduce false forced writes. To do this, look at the distribution of instance locks in GC_FILES_TO_LOCKS and check the data in the files.

Lock Names and Lock Formats

The following section describes the lock names and lock formats of locks. The topics in this section are:

- Lock Names and Lock Name Formats
- Lock Names
- Lock Types and Names

Lock Names and Lock Name Formats

Internally, Oracle global lock name formats used one of the following formats:

- `type ID1 ID2`
- `type, ID1, ID2`
- `type (ID1, ID2)`

Where:

`type`	A two-character type name for the lock type, for example, BL, TX, TM
`ID1`	The first lock identifier. The meaning and format of this identifier differs from one lock type to another.
`ID2`	The second lock identifier. The meaning and format of this identifier differs from one lock type to another.

For example, a space management lock might be named ST00. A lock might be named BL 1 900.

The clients of the lock manager define the lock type, for example *BL* for a lock, and two parameters, id1 and id2, and pass these parameters to the GCS API to open a lock. The lock manager does not distinguish between different types of locks. Each component of Oracle defines the type and the two parameters for its own needs, in other words, id1 and id2 have a meaning consistent with the requirements of each component.

Lock Names

All locks are Buffer Cache Management locks. Buffer Cache Management locks are of type *BL*. The syntax of lock names is `type ID1 ID2`, where:

`type` Is always BL because locks are buffer locks.

`ID1` The database address of the blocks.

`ID2` The block class.

Examples of lock names are:

`BL (100, 1)` This is a data block with lock element 100.

`BL (1000, 4)` This is a segment header block with lock element 1000.

`BL (27, 1)` This is an undo segment header with rollback segment #10. The formula for the rollback segment is 7 + (10 * 2).

Lock Types and Names

There are several different types and names of locks as shown in Table A–3:

Table A–3 Locks Types and Names

Type	Lock Name	Type	Lock Name
CF	Controlfile Transaction	PS	Parallel Execution Process Synchronization
CI	Cross-Instance Call Invocation	RT	Redo Thread
DF	Datafile	SC	System Commit Number
DL	Direct Loader Index Creation	SM	SMON
DM	Database Mount	SN	Sequence Number
DX	Distributed Recovery	SQ	Sequence Number Enqueue
FS	File Set	SV	Sequence Number Value
KK	Redo Log *Kick*	ST	Space Management Transaction
IN	Instance Number	TA	Transaction Recovery
IR	Instance Recovery	TM	DML Enqueue
IS	Instance State	TS	Temporary Segment (also Table-Space)

Table A–3 Locks Types and Names

Type	Lock Name	Type	Lock Name
MM	Mount Definition	TT	Temporary Table
MR	Media Recovery	TX	Transaction
IV	Library Cache Invalidation	UL	User-Defined Locks
L[A-P]	Library Cache Lock	UN	User Name
N[A-Z]	Library Cache Pin	WL	Begin written Redo Log
Q[A-Z]	Row Cache	XA	Instance Registration Attribute Lock
PF	Password File	XI	Instance Registration Lock
PR	Process Startup		

B

A Case Study in Real Application Clusters Database Design

This appendix describes a case study that presents a methodology for designing systems optimized for **Oracle Real Application Clusters**.

Case Study Overview

The case study presented in this appendix provides techniques for designing new applications for use with Real Application Clusters. You can also use these techniques to evaluate existing applications and determine how well suited they are for migration to Real Application Clusters.

Note: Always remember that your goal is to minimize contention: doing so results in optimized performance.

This case study assumes you have made an initial database design. To optimize your Real Application Clusters design, follow this methodology:

1. Develop an initial database design.
2. Analyze access to tables.
3. Analyze transaction volume.
4. Decide how to partition users and data.
5. Decide how to partition indexes, if necessary.
6. Implement and tune your design.

> **See Also:** Part II, "Scaling Applications and Designing Databases for Real Application Clusters", for detailed information on this methodology

Case Study: From Initial Database Design to Real Application Clusters

This case study is a practical demonstration of analytical techniques. Although your specific applications will differ from the example in this appendix, this case study should help you to understand the process. The topics in this section are:

Eddie Bean Catalog Sales

The case study is about the fictitious *Eddie Bean* catalog sales company. This company has many order entry clerks processing telephone orders for various products. Shipping clerks fill orders and accounts receivable clerks handle billing. Accounts payable clerks handle orders for supplies and services the company requires internally. Sales managers and financial analysts run reports on the data. This company's financial application has three business processes operating on a single database:

- Order entry
- Accounts payable
- Accounts receivable

Eddie Bean Database Tables

Tables from the Eddie Bean database include:

Table B–1 Eddie Bean Sample Tables

Table	Contents
`order_header`	Order number, customer name and address.
`order_item`	Products ordered, quantity, and price.
`organizations`	Names, addresses, phone numbers of customers and suppliers.
`accounts_payable`	Tracks the company's internal purchase orders and payments for supplies and services.
`budget`	Balance sheet of the company's expenses and income.
`forecasts`	Projects future sales and records current performance.

Eddie Bean Users

Various application users access the database to perform different functions:

- Order entry clerks
- Accounts payable clerks
- Accounts receivable clerks
- Shipping clerks
- Sales manager
- Financial analyst

The Eddie Bean Application Profile

Operation of the Eddie Bean application is fairly consistent throughout the day: order entry, order processing, and shipping occur all day. These functions are not for example, segregated into separate one-hour time slots.

About 500 orders are entered each day. Each order header is updated about 4 times during its lifetime. So we expect about 4 times as many updates as inserts. There are many selects, because many employees are querying order headers: people doing sales work, financial work, shipping, tracing the status of orders, and so on.

There are on average 4 items for each order. Order items are never updated: an item can be deleted and another item entered. The `order_header` table has four indexes. Each of the other tables has a primary key index only.

Budget and forecast activity has a much lower volume than the order tables. They are read frequently, but modified infrequently. Forecasts are updated more often than budgets, and are deleted once they go into actuals.

The vast bulk of the deletes are performed as a nightly batch job. This maintenance activity does not, therefore, need to be included in the analysis of normal functioning of the application.

Analyzing Access to Tables

Begin by analyzing the existing (or expected) access patterns for tables in your database. Then decide how to partition the tables and group them according to access pattern.

- Table Access Analysis Worksheet
- Case Study: Table Access Analysis

Table Access Analysis Worksheet

List all your high-activity database tables in a worksheet like the one shown in Table B–2:

Table B–2 Table Access Analysis Worksheet

	Daily Access Volume							
	Read Access		Write Access					
	Select		Insert		Update		Delete	
Table Name	Operations	I/Os	Operations	I/Os	Operations	I/Os	Operations	I/Os

To complete this worksheet, estimate the volume of each type of operations. Then calculate the number of reads and writes (I/Os) the operations entail.

Estimating Volume of Operations

For each type of operation to be performed on a table, enter a value reflecting *the normal volume you would expect in a day.*

Note: The emphasis throughout this analysis is on *relative values*—gross figures describing the normal use of an application. Even if an application does not yet exist, you can project the types of users and estimate relative levels of activity. Maintenance activity on the tables is not generally relevant to this analysis.

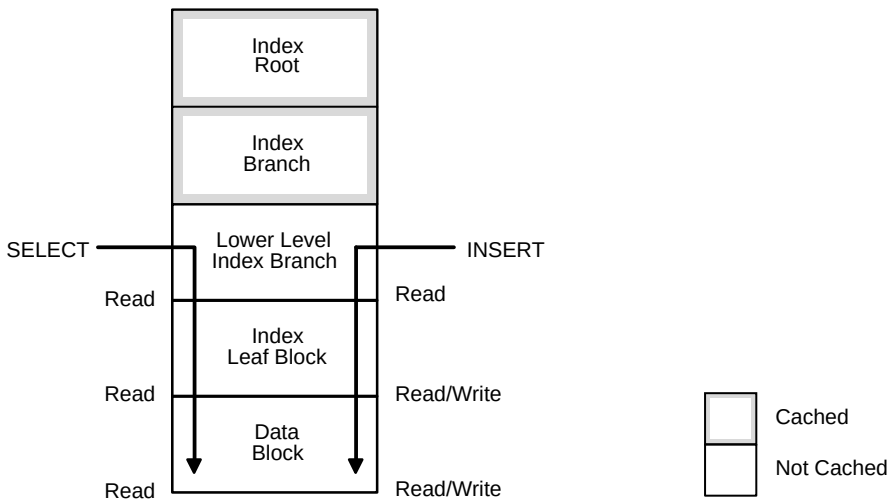
Calculating I/Os for Each Operation

For each value in the Operations column, calculate the number of I/Os that will be generated using a worst-case scenario. The SELECT operation involves read access, and the INSERT, UPDATE and DELETE operations involve both read and write access. These operations access not only data blocks, but also any related index blocks.

Note: The number of I/Os generated for each operation changes *by table* depending on the access path of the table, and the table's size. It also changes depending on the number of indexes a table has. A small index, for example, can have only a single index branch block.

For example, Figure B–1 illustrates read and write access to data in a large table in which two levels of the index are not in the buffer cache and only a high level index is cached in the **System Global Area (SGA)**.

Figure B–1 Number of I/Os for Each SELECT or INSERT Operation



In this example, assuming that you are accessing data by way of a primary key, a SELECT requires three I/Os:

1. One I/O to read the first lower level index block.
2. One I/O to read the second lower level index block.
3. One I/O to read the data block.

> **Note:** If all of the root and branch blocks are in the SGA, then a SELECT can entail only two I/Os: read leaf index block, read data block.

An INSERT or DELETE statement requires at least five I/Os:

1. One I/O to read the data block.
2. One I/O to write the data block.
3. Three I/Os *per index*: 2 to read the index entries and 1 to write the index.

One UPDATE in this example entails seven I/Os:

1. One I/O to read the first lower level index block.
2. One I/O to read the second lower level index block.
3. One I/O to read the data block.
4. One I/O to write the data block.
5. One I/O to read the first lower level index block again.
6. One I/O to read the second lower level index block again.
7. One I/O to write the index block.

> **Note:** An INSERT or DELETE affects *all* indexes, but an UPDATE sometimes affects only *one* index. Check the number of changed index keys.

I/Os for Each Operation for Sample Tables

In the case study, the number of I/Os for each operation differs from table to table because the number of indexes differs from table to table.

Table B–3 shows how many I/Os are generated by each type of operation on the `order_header` table. It assumes that the `order_header` table has four indexes.

Table B–3 Number of I/Os for each Operation: Sample ORDER_HEADER Table

Operation	SELECT	INSERT	UPDATE	DELETE
Type of Access	read	read / write	read / write	read / write
Number of I/Os	3	14	7	14

Note: Adjust these figures depending upon the actual number of indexes and access path for each table in your database.

Table B–4 shows how many I/Os generated for each operation for each of the other tables in the case study, assuming each of them has a primary key index only.

Table B–4 Number of I/Os for each Operation: Other Sample Tables

Operation	SELECT	INSERT	UPDATE	DELETE
Type of Access	read	read / write	read / write	read / write
Number of I/Os	3	5	7	5

For this analysis, you can disregard the fact that changes made to data also generate **rollback segment**, entailing additional I/Os. These I/Os are **instance**-based. Therefore, they should not cause problems with your Real Application Clusters applications.

See Also: *Oracle9i Database Concepts* for more information about indexes

Case Study: Table Access Analysis

Table B–5 shows rough figures reflecting normal use of the application in the case study.

Table B–5 Case Study: Table Access Analysis Worksheet

	Daily Access Volume							
	Read Access		Write Access					
	Select		Insert		Update		Delete	
Table Name	Operations	I/Os	Operations	I/Os	Operations	I/Os	Operations	I/Os
order_header	20,000	60,000	500	7,000	2,000	14,000	1,000	14,000
order_item	60,000	180,000	2,000	10,000	0	0	4,030	20,150
organizations	40,000	120,000	10	50	100	700	0	0
budget	300	900	1	5	2	14	0	0
forecasts	500	1,500	1	5	10	70	2	10
accounts_payable	230	690	50	250	20	140	0	0

You can make the following conclusions from the data in Table B–5:

- Only the order_header and order_item tables have significant levels of write access.
- organizations, by contrast, is predominantly a read-only table. While a certain number of INSERT, UPDATE, and DELETE operations will maintain it, its normal use is SELECT-only.

Analyzing Transaction Volume by Users

Begin by analyzing the existing (or expected) access patterns for tables in your database. Then partition the tables and group them according to access pattern.

- Transaction Volume Analysis Worksheet
- Case Study: Transaction Volume Analysis

Transaction Volume Analysis Worksheet

For each table with a high volume of write access, analyze the transaction volume per day for each type of user.

Note: For read-only tables, you do *not* need to analyze transaction volume by user type.

Use worksheets like the one in Table B–6:

Table B–6 Transaction Volume Analysis Worksheet

Table Name:									
		Daily Transaction Volume							
		Read Access		Write Access					
		Select		Insert		Update		Delete	
Type of User	No.Users	Operations	I/Os	Operations	I/Os	Operations	I/Os	Operations	I/Os

Begin by estimating the volume of transactions by each type of user and then calculate the number of I/Os required.

Case Study: Transaction Volume Analysis

The following tables show transaction volume analysis of the three tables in the case study that have high write access levels: `order_header`, `order_item`, and `accounts_payable`.

ORDER_HEADER Table

Table B–7 shows rough estimates for values in the `order_header` table in the case study.

Table B–7 Case Study: Transaction Volume Analysis: ORDER_HEADER Table

Table Name: ORDER_HEADER									
Type of User	**No. Users**	**Daily Transaction Volume**							
		Read Access		**Write Access**					
		Select		**Insert**		**Update**		**Delete**	
		Operations	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	5,000	15,000	500	7,000	0	0	0	0
Accounts payable clerk	5	0	0	0	0	0	0	0	0
Accounts receivable clerk	5	6,000	18,000	0	0	1,000	7,000	0	0
Shipping clerk	4	4,000	12,000	0	0	1,000	7,000	0	0
Sales manager	2	3,000	9,000	0	0	0	0	0	0
Financial analyst	2	2,000	6,000	0	0	0	0	0	0

You can make the following conclusions from the data in Table B–7:

- Order entry clerks perform all inserts on this table
- Accounts receivable and shipping clerks perform all updates
- Sales managers and financial analysts only perform select operations
- Accounts payable clerks never use the table

Deletes are performed as a maintenance operation, so you do not need to consider them in this analysis. Furthermore, the application developers realize that sales managers normally access data for the current month, whereas financial analysts access mostly historical data.

ORDER_ITEM Table

Table B–8 shows rough estimates for values in the `order_item` table in the case study.

Table B–8 Case Study: Transaction Volume Analysis: ORDER_ITEM Table

Table Name: ORDER_ITEM									
Type of User	**No. Users**	**Daily Transaction Volume**							
		Read Access		**Write Access**					
		Select		**Insert**		**Update**		**Delete**	
		Operations	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	15,000	45,000	2,000	10,000	0	0	20	100
Accounts payable clerk	5	0	0	0	0	0	0	0	0
Accounts receivable clerk	5	18,000	54,000	0	0	0	0	10	50
Shipping clerk	4	12,000	36,000	0	0	0	0	0	0
Sales manager	2	9,000	27,000	0	0	0	0	0	0
Financial analyst	2	6,000	18,000	0	0	0	0	0	0

The following conclusions can be drawn from Table B–8:

- Order entry clerks perform all inserts on this table
- Updates are rarely performed
- Accounts receivable clerks, shipping clerks, sales managers and financial analysts perform a heavy volume of select operations on the table
- Accounts payable clerks never use the table

The `order_header` table has more writes than `order_item` because the order header tends to require more changes of status, such as address changes, than the list of available products. The `order_item` table is seldom updated because new items are listed as journal entries.

ACCOUNTS_PAYABLE Table

Table B–9 shows rough figures for the `Accounts_payable` table in the case study. Although this table does not have a particularly high level of write access, we have analyzed it because it contains the main operation that the accounts payable clerks perform.

Table B–9 Case Study: Transaction Volume Analysis: ACCOUNTS_PAYABLE Table

Table Name: ACCOUNTS_PAYABLE									
		Daily Transaction Volume							
		Read Access		**Write Access**					
Type of	**No.**	**Select**		**Insert**		**Update**		**Delete**	
User	**Users**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	0	0	0	0	0	0	0	0
Accounts payable clerk	5	200	600	50	250	20	140	0	0
Accounts receivable clerk	5	0	0	0	0	0	0	0	0
Shipping clerk	4	0	0	0	0	0	0	0	0
Sales manager	2	0	0	0	0	0	0	0	0
Financial analyst	2	30	90	0	0	0	0	0	0

You can make the following conclusions from the data in this table:

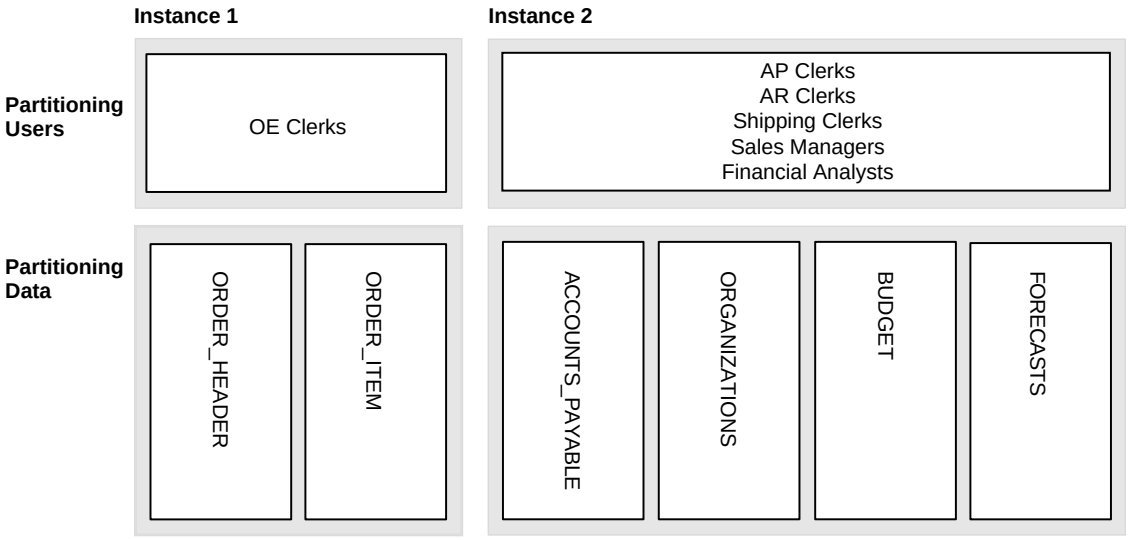
- Accounts payable clerks send about 50 purchase orders per day to suppliers. These clerks are the only users who change the data in this table.
- Financial analysts occasionally study the information.

Deletes are performed as a maintenance operation, so you do not need to consider them in this analysis.

Case Study: Initial Partitioning Plan

In the case study, the large number of order entry clerks doing heavy insert activity on the `order_header` and `order_item` tables should not be separated across machines. You should concentrate these users on one **node** along with the two tables they use most. A good starting point is to set aside one node for the Order Entry clerks, and one node for all other users as illustrated in Figure B–2.

Figure B–2 Case Study: Partitioning Users and Data



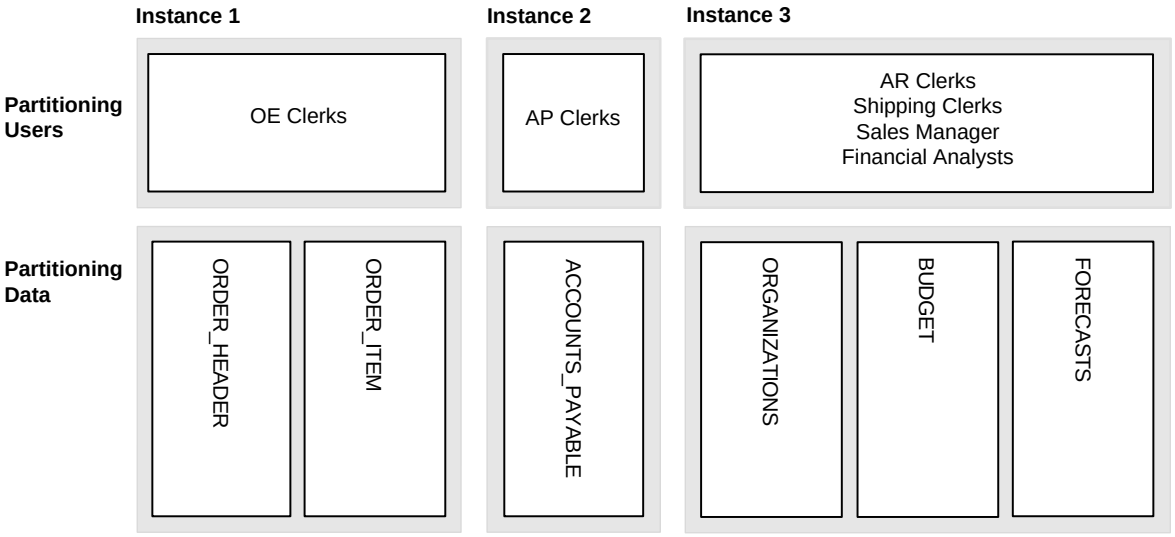
The system in Figure B–2 is probably well balanced across nodes. The database intensive reporting done by financial analysts takes a significant amount of system resources, whereas the transactions run by the order entry clerks are relatively simple.

Attempting to use **load balancing** by manipulating the number of users across the system is typically useful, but not always critical. Reducing contention has a more significant effect on tuning than implementing load balancing does.

Case Study: Further Partitioning Plans

In the case study it is also clear that accounts payable data is written exclusively by accounts payable clerks. You can thus effectively partition this data onto a separate **instance** as shown in Figure B–3.

Figure B–3 Case Study: Partitioning Users and Data: Design Option 1



When all users needing write access to a certain part of the data are concentrated on one node, the global enqueues all reside on that node. In this way, resource ownership does not move between instances. Based on this analysis, you have two design options as described under the following headings.

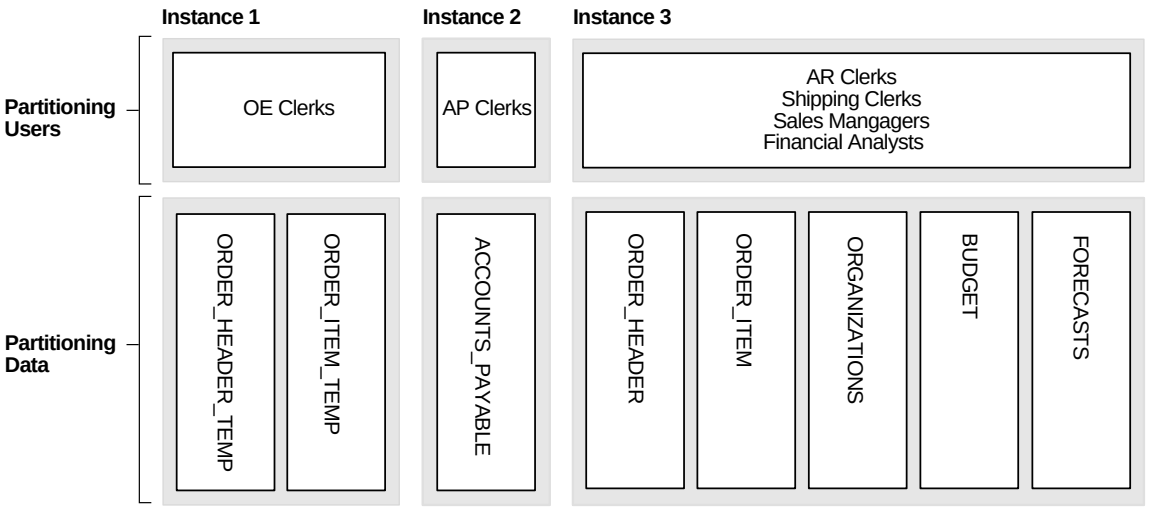
Design Option 1

You can set up your system as shown in Figure B–3 with all order entry clerks on one instance to minimize contention for exclusive GCS resources on the tables. This allows sales managers and financial analysts to get up-to-the-minute information. Since they do want data that is predominantly historical, there should not be too much contention for current records.

Design Option 2

Alternatively, you could implement a separate temporary table for `order_item/ order_header` as shown in Figure B–4. This table is only for recording new order information. Overnight, you could incorporate changes into the main table against which all queries are performed. This solution would work well if it is not required that financial analysis have current data. This is probably an acceptable solution only if they are primarily interested in looking at historical data. This would not be appropriate if the financial analysts needed up-to-the-minute data.

Figure B–4 Case Study: Partitioning Users and Data: Design Option 2



Partitioning Indexes

You need to consider index partitioning if multiple nodes in your system are inserting into the same index. In this situation, you must ensure that different instances insert into different points within the index.

Note: This problem is avoided in the Eddie Bean case study because application and data usage are partitioned.

Implement and Tune Your Design

Up to this point, you conducted an analysis using estimated figures. To finalize your design you must now either prototype the application or actually implement it. By observing the actual system, you can tune it further.

To do this, try the following techniques:

- Identify blocks that are being forced written and determine where contention exists.
- Consider moving users from one instance to another to reduce forced disk writes and false forced writes.
- If there are forced disk writes on inserts, then adjust the free lists or use multiple sequence number generators so that inserts occur in different parts of the index.

See Also: *Oracle9i Database Performance Guide and Reference*

Glossary

alert file

A file that contains information about error messages and exceptions that can occur during database operations. Each database instance maintains one alert file.

buffer busy due to global cache

Buffer busy due to global cache is a wait event that is signaled when a process has to wait for a block to become available because another process is obtaining a resource for this block.

buffer busy waits

Buffer busy waits is a wait event that is signaled when a process cannot get a buffer because another process is using the buffer at that moment.

cache convert waits

The cache convert waits per transactions statistic is the total number of waits for all up-convert operations, such as **global cache null to S**, **global cache null to X**, and **global cache S to X**.

Cache Fusion

Cache Fusion allows the direct transfer of data blocks between instances by way of an **interconnect** without causing forced writes to disk. That is, when one instance needs a current or consistent-read copy of a data block from another instance for a query or DML operation, the holding instance can transmit the block directly into the cache of the requesting instance.

cache open waits

The cache open waits per transactions statistic is the total number of waits for **global cache open S** and **global cache open X**.

cluster

A set of instances that typically run on different **node**s. Each instance coordinates with the others when accessing the shared database residing on disk.

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 membership across the cluster. The Cluster Manager also monitors process health. The **Lock Monitor Process (LMON)**, 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.

connection load balancing

A feature that balances the number of active connections among various instances and **shared server** dispatchers for the same service. Because of service registration's ability to register with remote listeners, a listener is always aware of all instances and dispatchers. This way, a listener can send an incoming client request for a specific service to the least loaded instance and least loaded dispatcher regardless of its location.

connect-time failover

A client connect request is forwarded to another listener if the first listener is not responding. Connect-time failover is enabled by **service registration**, because the listener knows whether an instance is up prior to attempting a connection.

consistent gets

Consistent gets are the number of buffers that are obtained in consistent read (CR) mode.

consistent read

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 the block, then the current version of the block can 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.

Console

The **Oracle Enterprise Manager** Console gives you a central point of control for the Oracle environment through an intuitive graphical user interface (GUI) that provides powerful and robust system management.

control file

A file that records the physical structure of a database and contains the database name, the names and locations of associated databases and online redo log files, the timestamp of the database creation, the current log sequence number, checkpoint information and various other records about the database's structure and health.

CR blocks received per transaction

The number of CR blocks shipped from the instance that has a block in exclusive access mode to the instance requesting a CR version of this block.

cr request retry

The cr request retry statistic is a wait that is incurred whenever Oracle re-submits a consistent read request when Oracle detects that the holding instance is no longer available.

data dependent routing

A method of routing data based on how the data is used within an application.

datafile

A file that contains the contents of logical database structures, such as tables and indexes. One or more datafiles form a logical unit of storage called a tablespace. A datafile can be associated with only one tablespace and only one database.

db block changes

Db block changes is a statistic that shows the number of current buffers obtained in exclusive mode for DML.

db block gets

db block gets is a statistic that shows the number of current buffers obtained for a read.

Database Writer (DBWn)

The DBWn processes are responsible for writing modified (dirty) buffers in the database buffer cache to disk.

DBWR cross-instance writes

DBWR cross-instance writes (also known as *forced writes*) are the number of writes that an instance has to perform to disk to make a previously exclusively held block available for another instance to read into its buffer cache. DBWR cross-instance writes are practically eliminated with Cache Fusion, unless you specify a value greater than 0 (zero) for the `GC_FILES_TO_LOCKS` parameter.

dedicated server

A server that requires a dedicated server process for each user process. There is one server process for each client. **Oracle Net** sends the address of an existing server process back to the client. The client then resends its connect request to the server address provided. Contrast this with the **shared server.**

degree of parallelism (DOP)

The degree of parallelism specifies the number of processes, or threads, used in parallel operations. Each parallel process or thread can use one or two parallel execution processes depending on the SQL statement's complexity.

DFS Lock Handles

DFS Lock Handles are pointers to global resources. To perform operations on global enqueue service resources, the process first needs to acquire a DFS handle.

disk affinity

Disk affinity is the relationship between data on a disk and the instance that needs to access it. True disk affinity is only available in shared nothing disk configurations. This enables you to partition tablespaces across disks such that each partition is accessed by one and only one instance. The instance accessing the data on that disk has disk affinity.

dispatcher

A process that enables many clients to connect to the same server without the need for a dedicated server process for each client. A dispatcher handles and directs multiple incoming network session requests to shared server processes. See also **shared server.**

flow control messages sent

The number of flow-control (nullreq and nullack) messages that are sent by the LMS process.

flow control messages received

The number of flow-control (nullreq and nullack) messages received by the LMD process.

forced disk writes

Forced disk writes refer to the forced writing of a data block to disk by one instance when the data block is requested by another instance for a DML operation. Forced Writes are practically eliminated in Oracle9*i* with Cache Fusion, but they remain relevant if you specify 1:1 or 1:n releasable or fixed resources with the `GC_FILES_TO_LOCKS` parameter. In this case, Cache Fusion is disabled.

global cache bg acks

Global cache bg acks is a wait event that only can occur during startup or shutdown of an instance when the LMS process finalizes its operations.

global cache busy

The global cache busy statistic is a wait event that occurs whenever a session has to wait for an ongoing operation on the resource to complete.

global cache cr cancel wait

The global cache cr cancel wait statistic is a wait event that occurs whenever a session waits for the AST to complete for a canceled block access request. Cancelling the request is part of the Cache Fusion Write Protocol.

global cache converts

Global cache converts are resource converts of buffer cache blocks. This statistic is incremented whenever GCS resources are converted from Null to Exclusive, Shared to Exclusive, or Null to Shared.

global cache convert time

Global cache convert time is the accumulated time that all sessions require to perform global conversions on GCS resources.

global cache convert timeouts

Global cache convert timeouts are incremented whenever a resource operation times out.

global cache cr block flush time

Global cache cr block flush time is the time waited for a log flush when a CR request is served. Once LGWR has completed flushing the changes to a buffer that is on the log flush queue, LMS can send it. It is part of the serve time.

global cache cr blocks received

When a process requests a consistent read for a data block that is not in its local cache, it sends a request to another instance. Once the request is complete, in other words, the buffer has been received, Oracle increments the statistic.

global cache cr block receive time

The global cache cr block receive time statistic records the total time required for consistent read requests to complete. In other words, it records the accumulated round-trip time for all requests for consistent read blocks.

global cache cr blocks served

The global cache cr blocks served statistic is the number of requests for a consistent read block served by LMS. Oracle increments this statistic when the block is sent.

global cache cr block build time

The global cache cr block build time statistic is the time that the LMS process requires to create a consistent read block on the holding instance

global cache cr block send time

The global cache cr block send time statistic is the time required by LMS to initiate a send of a consistent read block. For each request, timing begins when the block is sent and stops when the send has completed. This statistic only measures the time it takes to initiate the send; it does not measure the time elapsed before the block arrives at the requesting instance.

global cache cr cancel wait

Await event that occurs when a session waits for the acquisition interrupt to complete for a canceled CR request. Cancelling the CR request is part of the Cache Fusion write protocol.

global cache cr request

The global cache cr request statistic is a wait event that occurs whenever a process has to wait for a pending CR request to complete. The process waited for either shared access to a block to be granted before reading the block from disk into the cache, or it waited for the LMS of the holding instance to send the block.

global cache cr timeouts

The global cache cr timeouts statistic identifies a request for a consistent read block that has an excessive delay and that has timed out. This could be due to system performance problems, a slow **interconnect** network, or dropped network packets. The value of this statistic should always be 0 (zero).

global cache current block flush time

The global cache current block flush time statistic is the time it takes to flush the changes to a block to disk, otherwise known as a *forced log flush*, before the block is shipped to the requesting instance

global cache current block pin time

The global cache current block pin time statistic is the time it takes to the pin the current block before shipping it to the requesting instance. Pinning a block is necessary to disallow further changes to the block while it is prepared to be shipped to another instance.

global cache current blocks received

The global cache current blocks received statistic is the number of current blocks received from the holding instance over the **interconnect**.

global cache current block receive time

The global cache current block receive time statistic is the accumulated round-trip time for all requests for current blocks

global cache current block send time

The global cache current block send statistic is the time it takes to send the current block to the requesting instance over the **interconnect**.

global cache current blocks served

The global cache current blocks served statistic is the number of current blocks shipped to the requesting instance over the **interconnect**

global cache freelist wait

The global cache freelist wait statistic is a wait event that occurs when Oracle must wait after it detects that the local element free list is empty.

global cache freelist waits

The global cache freelist waits statistic is the number of times Oracle found the resource element free list empty.

global cache gets

The global cache gets statistic is the number of buffer gets that result in opening a new resource with the GCS.

global cache get time

The global cache get time statistic is the accumulated time of all sessions needed to open a GCS resource for a local buffer.

global cache initialization parameters

Global cache initialization parameters are initialization parameters that determine the size of the collection of global that protect the database buffers on all instances.

Note: Manually setting `GC_FILES_TO_LOCKS` overrides the default resource control behavior in Real Application Clusters.

---

global cache null to S

The global cache null to S statistic is a wait event that occurs whenever a session has to wait for a resource conversion to complete.

global cache null to X

The global cache null to X statistic is a wait event that occurs whenever a session has to wait for this resource conversion to complete.

global cache open S

The global cache open S statistic is a wait event that occurs when a session has to wait for receiving permission for shared access to the requested resource.

global cache open X

The global cache open X statistic is a wait event that occurs when a session has to wait for receiving a exclusive access to the requested resource.

global cache S to X

The global cache S to X statistic is a wait event that occurs whenever a session has to wait for this resource conversion to complete.

global cache pending ast

The global cache pending ast statistic is a wait event that can occur when a process waits for an acquisition interrupt before Oracle closes a resource element.

global cache pred cancel wait

A wait event that occurs when a session must wait for the acquisition interrupt to complete for a canceled predecessor read request. Cancelling a predecessor read request is part of the Cache Fusion write protocol.

global cache retry prepare

The global cache retry prepare statistic is a wait event that occurs whenever Oracle fails to prepare a buffer for a consistent read or Cache Fusion request, and when Oracle cannot ignore or skip this failure.

Global Cache Service (GCS)

The Global Cache Service is the process that implements Cache Fusion. It maintains block modes for blocks in the global role and is responsible for block transfers among instances. The Global Cache Service accomplishes these tasks using background processes such as the Global Cache Service process (LMS) and the Global Enqueue Service process (GES).

Global Cache Service Processes (LMSn)

The Global Cache Service Processes (LMSn) handle remote Global Cache Service messages. Current Real Application Clusters software provides for up to 10 Global Cache Service Processes. The number of LMSn processes varies depending on the amount of messaging traffic among **node**s in the cluster. The LMSn processes handle the acquisition interrupt and blocking interrupt requests from the remote instances for Global Cache Service resources. For cross-instance consistent read requests, LMSn creates a consistent read version of the block and sends it to the requesting instance. LMSn also controls the flow of messages to remote instances.

global database name

The global database name is the full name of the database that uniquely identifies it from another database. The global database name is of the form `database_name.database_domain`, for example, sales.us.acme.com.

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 resource requests. The LMD process also handles deadlock detection Global Enqueue Service requests. Remote resource requests are requests originating from another instance.

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 and Global Enqueue Services. In particular, LMON handles the part of recovery associated with global resources. LMON-provided services are also known as cluster group services (CGS).

global lock async converts

The global lock async converts statistic is the number of resources that Oracle converted from an incompatible mode.

global lock sync gets

The global lock sync gets statistic is the number of GCS resources that Oracle must open synchronously. Sync gets are mostly for GES resources (for example, library cache resources).

global lock async gets

The global lock async gets statistic is the number of GES resources that Oracle must open asynchronously. Async gets are only used for GES resources and include the number of global cache gets.

global lock get time

The global lock get time statistic is the accumulated time for all GES resources that Oracle needed to open.

global lock sync converts

The global lock sync converts statistic is the number of GES resources that Oracle converted from an incompatible mode. Sync converts occur mostly for GES resources.

global lock convert time

The global lock convert time statistic is the accumulated time for all global lock sync converts and global lock async converts.

high water mark

The high water mark is the highest limit within a segment for which space has been allocated to store data blocks. When a commit executes, if the new limit is greater than the previous limit, the high water mark is updated.

hybrid database

A hybrid database is one that has both OLTP and Data Warehouse processing characteristics.

initialization parameter file

The initialization parameter file is a file with parameter settings that initialize the database (`init`*`db_name`*`.ora`). In the case of Real Application Clusters, it initializes the instances within a cluster (`initsid.ora`). The default single initialization parameter file is known as `SPFILE.ORA`.

instance

For a Real Application Clusters database, each **node** within the cluster has an instance of the running Oracle9*i* software referencing the database.

When a database is started on a database server (regardless of the type of computer), 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 processes is called an *instance*. The memory and processes of an instance efficiently manage the database's data and serve the database users. You can connect to any instance to access information within a Real Application Clusters database.

Each instance has unique **Oracle System Identifier (SID)**, **instance name**, **instance number**, **rollback segments**, and **thread ID**.

instance groups

Use instance groups to limit the number of instances that participate in a parallel operation. You can create any number of instance groups, each consisting of one or more instances. You can then specify which instance group is to be used for any or all parallel operations. Parallel execution servers will only be used on instances that are members of the specified instance group.

instance name

Represents the name of the instance and is used to uniquely identify a specific instance when multiple instances share common service names. The instance name is identified by the `INSTANCE_NAME` parameter in the initialization parameter file. The instance name is identical to **Oracle System Identifier (SID)**.

instance number

A number that associates extents of data blocks with particular instances. The instance number enables you to start up an instance and ensure that it uses the

extents allocated to it for inserts and updates. This ensures that it does not use space allocated for other instances. The instance cannot use data blocks in another free list unless the instance is restarted with that instance number.

You can use various 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, `init`*`sid`*`.ora`.

interconnect

An interconnect is an arrangement of data paths that in the case of Real Application Clusters and Cache Fusion allows data to be sent between caches of disjoint **node**s

Inter-Process Communication (IPC)

The inter-process communication layer is an operating system-dependent component that enables transfers of messages, consistent-read, and current versions of data blocks between instances on different **node**s.

Lock Manager Servers (LMSn)

See **Global Cache Service Processes (LMSn)**.

listener

The listener process is a separate process residing on the server that listens for incoming client connection requests and manages server traffic. The listener brokers the client request, handing the request to the server when the server is available. 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.

Lock Manager Daemon Process (LMD)

See **Global Enqueue Service Daemon (LMD)**.

Lock Manager Server Process (LMS)

See **Global Cache Service Processes (LMSn)**.

Lock Monitor Process (LMON)

See **Global Enqueue Service Monitor (LMON)**.

load balancing

Load balancing is the even distribution of active database connections among instances. In the context of parallel execution, load balancing refers to the distribution of parallel execution server processes to spread work among the CPUs and memory resources.

lock buffers for read

The lock buffers for read statistic is the number of up-converts from Null to Shared.

lock gets per transaction

The lock gets per transaction statistic is the number of global lock sync gets and global lock async gets per transaction.

lock converts per transaction

The lock converts per transaction statistic is the number of global local sync converts and global lock async converts per transaction.

messages flow controlled

The number of messages intended to be sent directly but that are instead queued and delivered later by LMD / LMS.

messages received

The number of messages received by the LMD process.

messages sent directly

The number of messages sent directly by Oracle processes.

messages sent indirectly

The number of messages explicitly queued by Oracle processes.

Multi-threaded server (MTS)

See **shared server**.

Net8

See **Oracle Net**.

node

A node is machine where an instance resides.

operating system context switches

Operating system context switches occur when a thread's time allotment has elapsed, when a thread with a higher priority has become ready to run, or when a running thread needs to wait, for example, for I/O to complete.

operating system-dependent layer (OSD)

The operating system-dependent (OSD) layer is a software layer that consists of several software components developed either by vendors for UNIX platforms, or by Oracle for NT installations of the Oracle database. The OSD layer maps the key operating system/cluster-ware services required for operation of Real Application Clusters.

Oracle Data Gatherer

The Oracle Data Gatherer collects performance statistics for the **Oracle Performance Manager**. You must install the Oracle Data Gatherer on a **node** on your network.

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.

Oracle Enterprise Manager Console

The Oracle Enterprise Manager Console is a suite of GUI tools that make up the Oracle Enterprise Manager product.

Oracle Intelligent Agent

The Oracle Intelligent Agent is a process that runs on each of the **node** that functions as the executor of jobs and events sent by the console by way of the Management Server. The Oracle Intelligent Agent ensures high availability since the agent can function regardless of the status of the **Console** or network connections.

Oracle Net

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 Parallel Server Management

See Server Management.

Oracle Performance Manager

Oracle Performance Manager is 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.

Oracle Real Application Clusters

See **Real Application Clusters**.

Oracle System Identifier (SID)

An Oracle System Identifier is a name that identifies a specific instance of a running pre-release 8.1 Oracle database. 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 `init`*`db_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.

For pre-release 8.1 databases, `SID` identified the database. The `SID` was included in the part of the connect descriptor in a `tnsnames.ora` file, and in the definition of the network listener in the `listener.ora` file.

Oracle9*i* Enterprise Edition

Oracle9*i* Enterprise Edition is an object-relational database management system (ORDBMS). It provides the applications and files to manage a database. All other Real Application Clusters components are layered on top of the Oracle9*i* Enterprise Edition.

parallel automatic tuning

Parallel automatic tuning automatically controls values for all parameters related to parallel execution. These parameters affect several aspects of server processing, namely, the **degree of parallelism (DOP)**, the adaptive multi-user feature, and memory sizing. Initialize and automatically tune parallel execution by setting the initialization parameter `PARALLEL_AUTOMATIC_TUNING` to `true`.

parallel execution

Parallel execution refers to multiple processes operating together to complete a single database transaction. Parallel execution works on both single and multiple instance Oracle installations. Parallel execution is also referred to *parallel query.*

physical reads

The physical reads statistic is the number of disk reads that had to be performed when a request for a data block could not be satisfied from a local cache..

physical writes

The physical writes statistic is the number of write I/Os performed by the DBWn processes. This number includes the number of DBWR cross instance writes (forced writes) in Oracle9*i* when `GC_FILES_TO_LOCKS` is set. Setting `GC_FILES_TO_LOCKS` for a particular datafile will enable the use of the old ping protocol, and will not leverage the Cache Fusion architecture.

ping

Pings are actually **forced disk writes** which were common in previous Oracle cluster software products. Pings occurred because a data block can only be modified by one instance at a time. Before Real Application Clusters, if one instance modifies a data block that another instance requires, then whether a forced disk write occurs 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 a forced disk write to a block.

PMON process

PMON is 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 monitors dispatcher and server processes and restarts them if they have failed. As a part of **service registration**, PMON registers instance information with the listener.

raw devices

Raw devices are disks or partitions on disk drives that do not have a file system set up on them. Raw devices are used for Real Application Clusters since they enable the sharing of disks.

raw volumes

See **raw devices**.

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 Real Application Clusters scripts, initialization files, and datafiles to make the Oracle9*i* Enterprise Edition a Real Application Clusters database.

Recovery Manager (RMAN)

RMAN 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 (O/S) prompt or use the GUI-based Enterprise Manager Backup Manager.

redo log file

A redo log file is a file that contains a record of all changes made to data in the database buffer cache. If an instance failure occurs, then the redo log files are used to recover the modified data that was in memory.

remote instance undo block writes

The remote instance undo block writes statistic is the number of rollback segment undo blocks written to disk by DBWn as part of a forced write.

remote instance undo header writes

The remote instance undo header writes statistic is the number of rollback segment header blocks written to disk by DBWn as part of a forced write.

repository database

A repository database, such as that used by **Oracle Enterprise Manager,** is a set of tables in an Oracle database, to store data to manage Real Application Clusters environments. This database is separate from any shared Real Application Clusters database on the **node**s.

reverse key indexes

Reverse key indexes reverse the bytes of each column indexed while keeping the column order. This avoids performance degradation in Real Application Clusters where index modifications concentrate on a small set of leaf blocks. Reversing the keys of You cannot use reverse key indexes for index range scans.

rollback segment

Rollback segments contain transactions to undo changes to data blocks for uncommitted transactions. Rollback segments also 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 RBS*thread_id_rollback_number* by the `ROLLBACK_SEGMENTS` parameter in the instance initialization file.

seed database

A seed database is a preconfigured, ready-to-use database that requires minimal user input to create.

Server Management

Server Management (SRVM) is a comprehensive, integrated system management solution for managing Real Application Clusters environments. Server Management enables you to manage multi-instance databases in heterogeneous environments. Server Management is part of the open client/server architecture of **Oracle Enterprise Manager**. In addition to managing cluster databases, Server Management enables you to schedule jobs, perform event management, monitor performance, and obtain statistics to tune Real Application Clusters databases.

service name

A service name is a logical representation of a database, which 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 is the **global database name**, a name comprised of the database name (`DB_NAME`) and domain name (`DB_DOMAIN`), entered during installation or database creation.

If you are not sure what the global database name is, then you can obtain it from the combined values of the `SERVICE_NAMES` parameter in the initialization file.

service registration

Service registration is a feature by which the PMON process (or shared server Dispatcher processes when using shared server) automatically registers information with a listener. Because this information is registered with the listener, you do not need to configure the `listener.ora` file with this static information.

shared server

The shared server is a server configured to allow many user processes to share very few server processes. This means increases the number of users that can be supported. With shared server, 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. Contrast this with **dedicated server**.

star schemas

Star schemas are query-centric schemas that when represented in a diagram have a *fact* table at the center. The fact table usually contains the data element that is central to queries operating against the schema. A fact table is often quite large and is surrounded by several *dimension* tables that contain data that are attributes of the data in the fact table. Star schemas simplify query development because it is intuitive as to how to join attributes in the dimension tables with the fact table data. Star schemas are best suited for data warehousing environments and are thus less useful for OTLP environments.

striping

Striping refers to the interleaving of a related block of data across disks. If you properly implement striping, then it reduces I/O and improves performance. Because striping software is operating system-dependent, rely on your vendor documentation to ensure proper installation and configuration. There are two primary methods of striping, *single-user* or *multi-user*. These terms describe the type of environments in which each type of striping is most beneficial. The latter is commonly implemented for Real Application Clusters. With multi-user striping, the performance improvement is due to simultaneous disk arm movements reading related data on multiple hard drives. The degree to which average disk access time improves is proportional to the number of drives.

System Change Number (SCN)

System change numbers uniquely identify a committed transaction and the changes it makes. Within Real Application Clusters, system change numbers must not only be maintained within an instance, but they must also be synchronized across all instances with a cluster.

System Global Area (SGA)

The System Global Area is a group of shared memory structures that contain data and control information for an Oracle **instance**.

tablespace

A tablespace is a logical portion of an Oracle database used to allocate storage for table and index 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. For example, tablespaces commonly group all of an application's objects to simplify administrative operations.

Transmission Control Protocol/Interconnect Protocol (TCP/IP)

TCP/IP is a set of protocols that allow cooperating computers to share resources across a network.

thread ID

The thread ID is the number of a redo thread for an instance. Any available redo thread number can be used, but an instance cannot use the same thread number as another instance.

transaction monitor

A transaction monitor is a class of software products that provide a transaction execution layer above the operating system. Transaction monitors combine database updates and submit them to a database. In doing this, the transaction monitor manages some of the consistency and correctness of the database. The monitor ensures that the rules of transaction atomicity are adhered to; updates take place completely or not at all. The advantages of using transaction monitors include increased throughput.

transparent application failover (TAF)

Transparent application failover is a runtime failover mechanism for high-availability environments, such as Real Application Clusters and Oracle Real Application Clusters Guard, that refers to the failover and re-establishment 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.

User Datagram Protocol (UDP)

The User Datagram Protocol is a similar protocol to TCP/IP, however, it is simpler to administer. It is considered less reliable than TCP/IP because, for example, it does not guarantee message ordering.

User-mode IPC

User-mode IPC (Inter-process Communication) is an IPC-based protocol that directly accesses network hardware. As opposed to kernel-mode IPC, with

user-mode IPC the protocol avoids the overhead of copying data into kernel space, making system calls, and incurring context switches.

Virtual Interface Architecture (VIA)

Virtual Interface Architecture is an implementation of user mode IPC.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W