Oracle8*i* Parallel Server

Concepts

Release 2 (8.1.6)

December 1999

Part No. A76968-01

ORACLE

Oracle8*i* Parallel Server Concepts, Release 2 (8.1.6)

Part No. A76968-01

Primary Author: Mark Bauer.

Primary Contributors: Wilson Chan, Sohan Demel, Merrill Holt, and Michael Zoll.

Contributors: Christina Anonuevo, Lance Ashdown, David Austin, Bill Bridge, Sandra Cheever, Carol Colrain, Mark Coyle, Connie Dialeris, Karl Dias, Anurag Gupta, Deepak Gupta, Mike Hartstein, Andrew Holdsworth, Ken Jacobs, Ashok Joshi, Jonathan Klein, Jan Klokkers, Boris Klots, Anjo Kolk, Tirthankar Lahiri, Bill Lee, Lefty Leverenz, Juan Loaiza, Sajjad Masud, Neil Macnaughton, Ravi Mirchandaney, Rita Moran, Kant Patel, Erik Peterson, Mark Porter, Darryl Presley, Brian Quigley, Ann Rhee, Pat Ritto, Roger Sanders, Hari Sankar, Ekrem Soylemez, Vinay Srihari, Bob Thome, Alex Tsukerman, Tak Wang, Graham Wood, and Betty Wu.

Graphic Designer: Valarie Moore.

Contents

8 Scalability and Oracle Parallel Server

9 High Availability and Oracle Parallel Server

Part IV Reference

A Differences Between Releases

Send Us Your Comments

***Oracle8i Parallel Server Concepts*, Release 2 (8.1.6)**

Part No. A76968-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. 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 about this manual?

If you find any errors or have any other suggestions for improvement, please indicate 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) 506-7228 Attn: Oracle Server Documentation
- Postal service:
  Oracle Corporation
  Server Documentation Manager
  500 Oracle Parkway, 4OP12
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, and telephone number below.

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

Preface

This manual prepares you to successfully implement parallel processing by presenting Oracle Parallel Server concepts. Information in this manual applies to Oracle Parallel Server as it runs on all operating systems.

You should read this manual before reading the *Oracle8i Parallel Server Setup and Configuration Guide* and *Oracle8i Parallel Server Administration, Deployment, and Performance*. For general information about Oracle and administering the Oracle Server, refer to *Oracle8i Concepts*. and the *Oracle8i Administrator's Guide.*

> **See Also:** You can also use the *Oracle8i Parallel Server Documentation Online Roadmap* to help you use the online Oracle Parallel Server Documentation set.

What's New in Oracle8*i*?

This book has been re-written for Oracle8*i*. Oracle8*i* introduces Cache Fusion, a feature that reduces the overhead of resolving read / write conflicts caused by inter-instance contention. This greatly enhances performance as well as Oracle Parallel Server scalability.

> **See Also:** Appendix A, "Differences Between Releases" for information on feature changes from one release of Oracle Parallel Server to another.

Release 8.1.5

Release 8.1.5 introduced the first phase of Cache Fusion.

Release 8.1.6

Release 8.1.6 introduces further enhancements to Cache Fusion as well as the Primary / Secondary Instance feature. There are also several new performance statistics.

Intended Audience

This manual is written for database administrators and application developers who work with Oracle Parallel Server.

How this Book is Organized

This book presents Oracle Parallel Server concepts in four parts. It begins by describing parallel processing fundamentals for Oracle Parallel Server. The book then covers synchronization processing among instances and explains the fundamentals of how Oracle Parallel Server is implemented. It ends with reference material that includes an appendix describing the differences between versions, and an appendix describing the implementation restrictions of Oracle Parallel Server.

Structure

This book has been reorganized for 8.1.6 into the following four parts:

Part I, "Parallel Processing Fundamentals"

Chapter 1, "Introduction to Oracle Parallel Server"	This chapter introduces parallel processing and parallel database technologies that offer advantages for online transaction processing and decision support applications.
Chapter 2, "Parallel Hardware Architecture"	This chapter describes the hardware components and high-level architectural models that typify cluster environments.
Chapter 3, "Oracle Parallel Server Architecture"	This chapter describes the architectural components that Oracle provides for Parallel Server processing that are in addition to the single-instance components.

Part II, "Oracle Parallel Server Lock Processing"

Chapter 4, "Inter-Instance Coordination"	This chapter provides a detailed discussion of the inter-instance coordination activities involved in a cluster.
Chapter 5, "Parallel Cache Management"	This chapter provides detail on Parallel Cache Management locking.

Part III, "Implementing Oracle Parallel Server"

Chapter 6, "Oracle Parallel Server Components"	This chapter describes the implementation components for Oracle Parallel Server applications.
Chapter 7, "Oracle Parallel Server Storage Considerations"	This chapter describes the storage considerations for Oracle Parallel Server applications.
Chapter 8, "Scalability and Oracle Parallel Server"	This chapter describes the scalability features of Oracle Parallel Server.
Chapter 9, "High Availability and Oracle Parallel Server"	This chapter describes the concepts and some of the "best practices" methodologies for using Oracle Parallel Server to implement high availability.

Part IV, "Reference"

Appendix A, "Differences Between Releases"	This appendix describes the differences between this release and previous releases of Oracle that pertain to Oracle Parallel Server.
Appendix B, "Restrictions"	This appendix lists restrictions for Oracle Parallel Server.

Related Documents

After reading this manual, read *Oracle8i Parallel Server Setup and Configuration Guide* and the *Oracle8i Parallel Server Administration, Deployment, and Performance.*

Read the following manuals for more information:

Installation Guides

- *Oracle8i Installation Guide* for Sun Solaris, HP 9000 and AIX-based systems
- *Oracle8i Installation Guide for Windows NT*
- *Oracle Diagnostics Pack Installation*

Operating System-Specific Administrative Guides

- *Oracle8i Administrator's Reference* for Sun Solaris, HP 9000 or AIX-based systems
- *Oracle Parallel Server Administrator's Guide for Windows NT*
- *Oracle8i Administrator's Guide for Windows NT*

Oracle Parallel Server Management

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

Oracle Server Documentation

- *Getting to Know Oracle8i*
- *Oracle8i Concepts*
- *Oracle8i Administrator's Guide*
- *Oracle8i Reference*
- *Net8 Administrator's Guide*

Conventions

This section explains the conventions used in this manual including the following:

- Text
- Syntax diagrams and notation
- Code examples

Text

This section explains the conventions used within the text:

UPPERCASE Characters

Uppercase text is used to call attention to command keywords, object names, parameters, filenames, and so on.

For example, "If you create a private rollback segment, the name must be included in the ROLLBACK_SEGMENTS parameter of the parameter file."

Italicized Characters

Italicized words within text are book titles or emphasized words.

Syntax Diagrams and Notation

The syntax diagrams and notation in this manual show the syntax for SQL commands, functions, hints, and other elements. This section tells you how to read syntax diagrams and examples and write SQL statements based on them.

Keywords

Keywords are words that have special meanings in the SQL language. In the syntax diagrams in this manual, keywords appear in uppercase. You must use keywords in your SQL statements exactly as they appear in the syntax diagram, except that they can be either uppercase or lowercase. For example, you must use the CREATE keyword to begin your CREATE TABLE statements just as it appears in the CREATE TABLE syntax diagram.

Parameters

Parameters act as place holders in syntax diagrams. They appear in lowercase. Parameters are usually names of database objects, Oracle datatype names, or expressions. When you see a parameter in a syntax diagram, substitute an object or expression of the appropriate type in your SQL statement. For example, to write a CREATE TABLE statement, use the name of the table you want to create, such as EMP, in place of the *table* parameter in the syntax diagram. (Note that parameter names appear in italics in the text.)

This list shows parameters that appear in the syntax diagrams in this manual and examples of the values you might substitute for them in your statements:

Parameter	Description	Examples
table	The substitution value must be the name of an object of the type specified by the parameter.	emp
'text'	The substitution value must be a character literal in single quotes.	'Employee Records'
condition	The substitution value must be a condition that evaluates to TRUE or FALSE.	ename > 'A'
date *d*	The substitution value must be a date constant or an expression of DATE datatype.	TO_DATE ('01-Jan-1996', DD-MON-YYYY')
expr	The substitution value can be an expression of any datatype.	sal + 1000
integer	The substitution value must be an integer.	72
subquery	The substitution value must be a SELECT statement contained in another SQL statement.	SELECT ename FROM emp
statement_name *block_name*	The substitution value must be an identifier for a SQL statement or PL/SQL block.	s1 b1

Code Examples

SQL and SQL*Plus commands and statements appear separated from the text of paragraphs in a monospaced font. For example:

```
INSERT INTO emp (empno, ename) VALUES (1000, 'SMITH');
ALTER TABLESPACE users ADD DATAFILE 'users2.ora' SIZE 50K;
```

Example statements may include punctuation, such as commas or quotation marks. All punctuation in example statements is required. All example statements terminate with a semicolon (;). Depending on the application, a semicolon or other terminator may or may not be required to end a statement.

Uppercase words in example statements indicate the keywords within Oracle SQL. When you issue statements, however, keywords are not case sensitive.

Lowercase words in example statements indicate words supplied only for the context of the example. For example, lowercase words may indicate the name of a table, column, or file.

Part I

Parallel Processing Fundamentals

Part I describes the fundamentals of Oracle Parallel Server processing. The chapters in this part are:

- Chapter 1, "Introduction to Oracle Parallel Server"
- Chapter 2, "Parallel Hardware Architecture"
- Chapter 3, "Oracle Parallel Server Architecture"

1

Introduction to Oracle Parallel Server

This chapter introduces parallel processing and the parallel database technology features of Oracle Parallel Server. Oracle Parallel Server offers significant advantages for Online Transaction Processing (OLTP), Electronic Commerce, Decision Support Systems (DSS), and hybrid system types. With the functionality of Oracle Parallel Server, such systems can effectively exploit the redundancy of parallel environments.

You can also use Oracle Parallel Server to deliver high performance, throughput, and high availability. Whatever your goal, your challenge is to successfully deploy these technologies to take full advantage of their multiprocessing powers. To do this, you must understand how Oracle Parallel Server works, what resources it requires, and how to effectively use it.

This chapter includes the following topics:

Note: Oracle Parallel Server is not the same as the parallel execution feature. Oracle Parallel Server refers to multiple computers accessing a shared database while parallel execution refers to multiple processes executing an operation in parallel. Parallel execution is available on both single-instance and Oracle Parallel Server installations.

See Also: *Oracle8i Concepts* for additional information on Oracle Parallel Server.

What Is Oracle Parallel Server?

Oracle Parallel Server is a robust computing environment that harnesses the processing power of multiple, interconnected computers. Oracle Parallel Server software and a collection of hardware known as a "cluster", unites the processing power of each component to become a single, robust computing environment. A cluster generally comprises two or more computers, or "nodes".

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

Harnessing the power of multiple nodes offers obvious advantages. If you divide a large task into sub-tasks and distribute the sub-tasks among multiple nodes, you can complete the task faster than if only one node did the work. This type of parallel processing is clearly more efficient than sequential processing. It also provides increased performance to process larger workloads and accommodate growing user populations.

If you establish high node-to-data affinity with accurate partitioning, you can effectively scale your applications to meet increasing data processing demands. As you add resources, Oracle Parallel Server can exploit them and extend their processing powers beyond the limits of the individual components.

You can use Oracle Parallel Server for many system types. For example, data warehousing applications accessing read-only data are prime candidates for Oracle Parallel Server. In addition, Oracle Parallel Server successfully manages increasing numbers of online transaction processing systems as well as hybrid systems that combine the characteristics of both read-only and read / write applications.

Oracle Parallel Server also serves as an important component of robust High Availability solutions. A properly configured Oracle Parallel Server environment can tolerate failures with minimal or no downtime.

Benefits of Oracle Parallel Server

Some of the most important benefits beyond the obvious advantages of parallel processing are described in the following sections. These benefits include improved throughput and scalability over single-instance systems and improved response time. An Oracle Parallel Server also provides an ideal High Availability solution by resolving node failure in a clustered environment.

Oracle Parallel Server environments are functionally transparent when compared to single-instance environments because they are functionally identical to single-instance environments.

Scalability

Scalability is the ability to add additional nodes to properly deployed Oracle Parallel Server applications and achieve markedly improved performance. Oracle Parallel Server can take advantage of additional equipment and harness the processing power of multiple systems.

High Availability

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

Transparency

Transparency is the functional equivalent of single-instance exclusive Oracle and shared configurations that use Oracle Parallel Server. Applications that run on single instance Oracle execute with the same results using Oracle Parallel Server. An Oracle database can be configured to execute in three different modes:

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

Installation of the Oracle Parallel Server option is required if you want to execute transactions from multiple nodes in shared mode. Oracle Parallel Server offers many performance features beyond those available in a single instance environment.

High Performance Features of Oracle Parallel Server

Oracle Parallel Server takes advantage of the parallel processing in a computer cluster without sacrificing Oracle's inherent transaction processing features. The following sections discuss certain features in Oracle, both in exclusive and shared modes, that result in improved application performance when these applications run using Oracle Parallel Server.

Buffer Cache Management Within a single instance, Oracle stores resources, such as data block and lock information, in a buffer cache that resides in memory. Storing this information locally reduces the amount of disk I/O necessary for database operations. Since each node in the Parallel Server has its own memory that is not shared with other nodes, Oracle Parallel Server must coordinate the buffer caches of different nodes while minimizing additional disk I/O that could reduce performance. The Oracle parallel cache management technology maintains the high-performance features of Oracle while coordinating multiple buffer caches.

> **See Also:** *Oracle8i Concepts* for detailed information about the buffer cache.

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

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

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

- When the blocks have not been used recently and new data requires buffer cache space (in shared or exclusive mode)
- During checkpoints (shared or exclusive mode)
- When another instance needs the blocks (only in shared mode)

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

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

If you operate Oracle in ARCHIVELOG mode, online redo log files are archived before they are overwritten. In Oracle Parallel Server, each instance can automatically archive its own redo log files or one or more instances can manually archive the redo log files for all instances.

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

2

Parallel Hardware Architecture

This chapter describes the hardware components and various high-level architectural models that typify cluster environments. The model you select to deploy your Oracle Parallel Server application depends on your processing goals.

Oracle Parallel Server environments are typically deployed with several nodes interconnected to form a cluster. This chapter explains the basic hardware for nodes as well as the hardware that is used to make the nodes into a cluster.

Topics in this chapter include:

Overview of Cluster Hardware Components

A cluster comprises two or more nodes that are linked by an interconnect. The interconnect serves as the communication path between the nodes in the cluster. The nodes use the interconnect for communication required to synchronize each instance's manipulation of the shared data. The shared data that the nodes access resides in storage devices. A cluster is also known as a "loosely coupled computer system".

The following sections describe these components in more detail.

What is a Node?

A node has four main components:

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

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

All clusters use CPUs in more or less the same manner. However, the remaining components, memory, storage, and the interconnect, can be configured in different ways for different purposes. The remaining sections of this chapter explain how clusters use these components by describing:

- Memory Access
- The High Speed Interconnect
- Clusters - Nodes and the Interconnect
- Storage Access in Clustered Systems

Memory Access

Multiple CPUs are typically configured to share main memory. This allows you to create a single computer system that delivers scalable performance. This type of

system is also less expensive to build than a single CPU with equivalent processing power. A computer with a single CPU is known as a "uniprocessor".

There are two configurations of shared memory systems:

- Uniform Memory Access
- Non-Uniform Memory Access

Shared memory systems are also known as "tightly coupled computer systems".

Uniform Memory Access

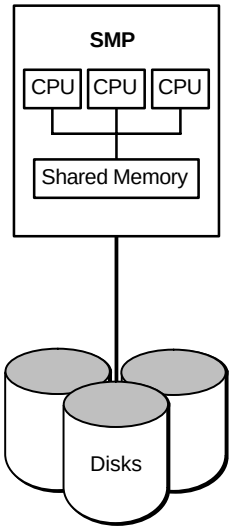
In uniform memory access configurations, or UMA, all processors can access main memory at the same speed. In this configuration, memory access is uniform. This configuration is also known as a Symmetric Multi-Processing system or "SMP".

Non-Uniform Memory Access

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

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

Figure 2–1 Tightly Coupled Shared Memory System or SMP/UMA



Advantages of Shared Memory

The parallel processing advantages of shared memory systems are:

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

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

The High Speed Interconnect

This is a high bandwidth, low latency communication facility that connects each node to the other nodes in the cluster. The high speed interconnect routes messages and other parallel processing-specific traffic among the nodes to coordinate each node's access to the data and to the data-dependent resources.

Oracle Parallel Server also makes use of user-mode interprocess communication (IPC) and "memory-mapped IPC". These substantially reduce CPU consumption and reduce IPC latency.

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

Clusters - Nodes and the Interconnect

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

Storage Access in Clustered Systems

Clustered systems use several architectural models. Each architecture uses a particular resource sharing scheme that is best used for a particular purpose.

This section describes the following architectures:

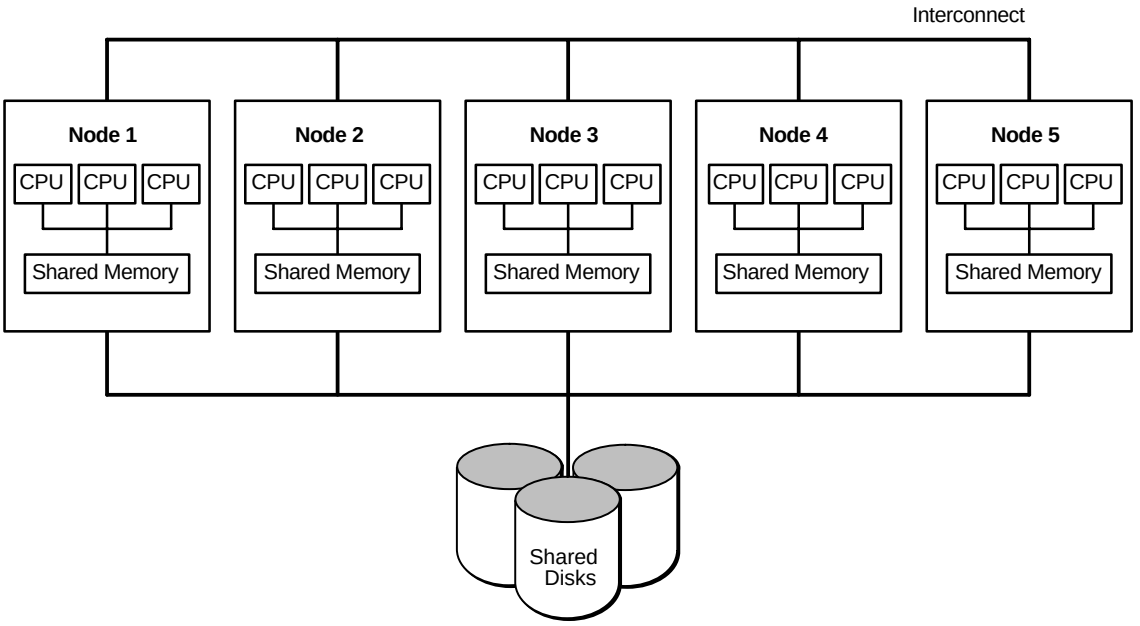
- Uniform Disk Access
- Non-Uniform Disk Access

This type of storage access is independent of the type of memory access. For example, a cluster of SMP nodes may be configured with either uniform or non-uniform disk subsystems.

Uniform Disk Access

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

Figure 2–2 Uniform Access Shared Disk System



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

The advantages of using parallel processing on shared disk systems are:

- Shared disk systems permit high availability; all data is accessible even if one node fails
- Shared disk systems provide incremental growth

Non-Uniform Disk Access

In some systems, the disk storage is attached to only one node. For that node, the access is local. For all other nodes, a request for disk access as well as the data must be forward by a software virtual disk layer over the interconnect to the node where the disk is locally attached. This means that the cost of a disk read or write varies significantly depending on whether the access is local or remote. The costs associated with reading or writing the blocks from the remote disks, including the

interconnect latency and the IPC overhead, all contribute to the increased cost of this type of operation versus the cost of the same type of operation using a uniform disk access configurations.

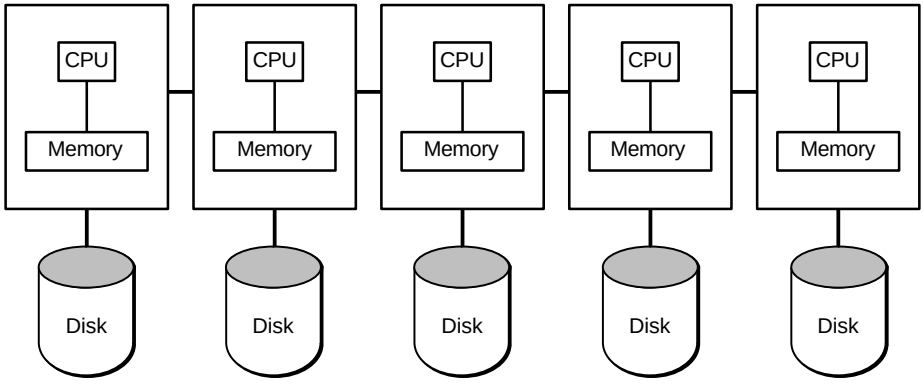
Non-uniform disk access configurations are commonly used on systems known as "shared nothing systems" or "Massively Parallel Processing (MPP) systems". For high availability, if a node fails, its local disks can usually be reconfigured to be local to another node. For these non-uniform disk access systems,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

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

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

Figure 2–3 illustrates a shared nothing system:

Figure 2–3 Non-uniform Disk Access



Oracle Parallel Server Runs on A Wide Variety of Clusters

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

3

Oracle Parallel Server Architecture

This chapter describes the architectural components that Oracle provides for Oracle Parallel Server processing. These are the components that are in addition to the components for single-instances; they are thus unique to Oracle Parallel Server. Some of these components are supplied with the Oracle software and others are vendor-specific.

Topics in this chapter include:

- Oracle Parallel Server Components for Clustered Systems
- The Cluster Manager
- Distributed Lock Manager
- The Cluster Interconnect and Inter-Process Communication (Node-to-Node)
- Disk Subsystems

> **See Also:** *Oracle8i Concepts* for more information about Oracle's single-instance architectural components.

Oracle Parallel Server Components for Clustered Systems

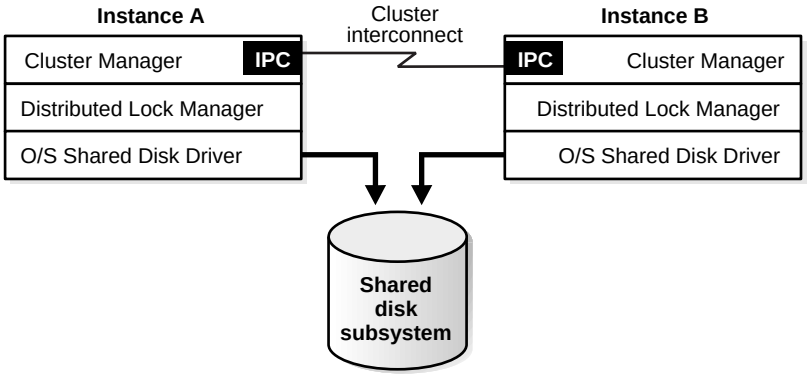
Based on the architectural models described in Chapter 2, the following explains the software required for implementing Oracle Parallel Server.

Each hardware vendor implements parallel processing using operating system dependent layers. These layers serve as communication links between the operating system and the Oracle Parallel Server software described in this chapter.

Overview of Components for Clustered Systems

A high-level view of these components appears in Figure 3–1.

Figure 3–1 Cluster Components for Parallel Processing



The Cluster Manager software oversees internode messaging that travels over the interconnect to coordinate internode operations. The Distributed Lock Manager oversees the operation of Parallel Cache Management functions. The following describes the following in more detail:

- The Cluster Manager
- Distributed Lock Manager
- The Cluster Interconnect and Inter-Process Communication (Node-to-Node)
- Disk Subsystems

The Cluster Manager

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

Oracle Parallel Server also cooperates with the Cluster Manager to achieve high availability. The Cluster Manager automatically starts and stops when the instance starts and stops.

Failure Detection

A Cluster Manager disconnect can occur for any of three reasons: the client disconnects voluntarily, the client's process terminates, or the client's node shuts down or fails. This is true even if one or more nodes fail. If the Cluster Manager determines that a node is inactive or not functioning properly, the Cluster Manager terminates all processes on that node or instance.

If there is a failure, recovery is transparent to user applications. The Cluster Manager automatically reconfigures the system to isolate the failed node and then notifies the Distributed Lock Manager of the status. Oracle Parallel Server then recovers the database to a valid state.

The Node Monitor

The Cluster Manager has a subset of functionality known as the "Node Monitor". The Node Monitor polls the status of various resources in a cluster including nodes, interconnect hardware and software, shared disks, and Oracle instances. The means by which the Cluster Manager and its Node Monitor performs these operations is based on Oracle's implementation of the operating system dependent layer.

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

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

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

> **See Also:** For more information on High Availability, please refer to Chapter 9.

Distributed Lock Manager

The Distributed Lock Manager is an integrated component of Parallel Server that coordinates simultaneous access to the shared database and to shared resources within the database. It does this to maintain consistency and data integrity. This section describes the following features of the Distributed Lock Manager:

- Transparency
- Distributed Architecture
- Fault Tolerance
- Resource Mastering
- Deadlock Detection
- Persistent Resources

Transparency

The coordination of access to resources that is performed by the Distributed Lock Manager is transparent to applications. Applications continue to use the same locking mechanisms as are used by the single instance environment.

Distributed Architecture

The Distributed Lock Manager maintains a lock database to record information about resources and locks held on these resources. This lock database resides in memory and is distributed throughout the cluster to all nodes. In this distributed architecture, each node participates in global lock management and manages a portion of the global lock database. This distributed lock management scheme provides fault tolerance and enhanced runtime performance.

Fault Tolerance

The Distributed Lock Manager is fault tolerant in that it provides continual service and maintains the integrity of the lock database even if multiple nodes fail. The shared database is accessible as long as at least one instance is active on that database after recovery completes.

Fault tolerance also enables instances within an Oracle Parallel Server to be started and stopped at any time, in any order. However, instance reconfiguration may cause a brief delay.

Resource Mastering

The Distributed Lock Manager maintains information about locks on all nodes that need access to a particular resource. The Distributed Lock Manager usually nominates one node to manage all information about a resource and its locks.

Oracle Parallel Server uses a static hashing lock mastering scheme. This mastering process hashes the resource name to one of the Parallel Server instances that acts as the master for the resource. This results in an even, arbitrary distribution of resources among all available nodes. Every resource is associated with a master node.

The Distributed Lock Manager optimizes the method of lock mastering used in each situation. The method of lock mastering affects system performance during normal runtime activity as well as during instance startup. Performance is optimized when a resource is mastered locally.

Deadlock Detection

The Distributed Lock Manager performs deadlock detection to all deadlock sensitive locks and resources. It does not control access to tables or objects in the database itself. Oracle Parallel Server uses the Distributed Lock Manager to coordinate concurrent access across multiple instances to resources such as data blocks and rollback segments.

Persistent Resources

The Distributed Lock Manager provides persistent resources. Resources maintain their state even if all processes or groups holding a lock on it have died abnormally.

Example of DLM Processing

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

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

Interaction with the Cluster Manager

The Distributed Lock Manager operates independently of the Cluster Manager. The Distributed Lock Manager relies on the Cluster Manager for timely and correct information about the status of other nodes. If the Distributed Lock Manager cannot get the information it needs from a particular instance in the cluster, it shuts down the instance. This ensures the integrity of Oracle Parallel Server databases, as each instance must be aware of all other instances to coordinate disk access.

The Cluster Interconnect and Inter-Process Communication (Node-to-Node)

Oracle Parallel Server derives most of its functional benefits from its ability to run on multiple interconnected machines. Oracle Parallel Server relies heavily on the underlying Inter-Process Communication (IPC) component to facilitate this.

The IPC defines the protocols and interfaces required for the Oracle Parallel Server environment to transfer messages between instances. Messages are the fundamental units of communication in this interface. The core IPC functionality is built around an asynchronous, queued messaging model. IPC is designed to send and receive discrete messages as fast as the hardware allows. With an optimized communication layer, various services can be implemented above it. This is how the Distributed Lock Manager performs its communication duties.

Disk Subsystems

In addition to the operating system dependent layers, Oracle Parallel Server also requires that all nodes must have simultaneous access to the disks. This gives multiple instances concurrent access to the same database.

Part II

Oracle Parallel Server Lock Processing

Part II describes the lock processing that Oracle Parallel Server performs to synchronize data access and ensure data integrity. The chapters in this part are:

- Chapter 4, "Inter-Instance Coordination"
- Chapter 5, "Parallel Cache Management"

4

Inter-Instance Coordination

This chapter provides a detailed discussion of the inter-instance coordination activities that take place in a cluster. As mentioned in Chapter 3, Oracle uses locks within a cluster to coordinate lock resources, data, and inter-instance data requests. This chapter describes the details about how Oracle coordinates these resources.

Topics in this chapter include:

- Synchronization
- Local Locks
- Global Locks
- Non-Parallel Cache Management Coordination
- Parallel Cache Management Coordination

Synchronization

Coordination of concurrent tasks within a cluster is called synchronization. Resources such as data blocks and locks must be synchronized as nodes within a cluster acquire and release ownership of them. The synchronization provided by Oracle Parallel Server maintains cluster-wide concurrency of the resources and in turn ensures the integrity of the shared data.

The key to successful parallel processing is to divide the tasks that require resources among the nodes so that very little synchronization is necessary. The less synchronization that is necessary, the better your system's speedup and scaleup. The overhead of synchronization can be very expensive if excessive inter-node communication is necessary.

The synchronization effort to achieve parallel processing among nodes ideally uses a high-speed interconnect linking the parallel processors. For parallel processing within a node, messaging is not necessary; shared memory is used instead. As mentioned in the previous chapter, messaging and locking between nodes is handled by the Distributed Lock Manager.

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

Each instance of Oracle Parallel Server has a dictionary cache, or row cache, containing data dictionary information in its System Global Area. The data dictionary structure is the same for Oracle instances in Oracle Parallel Server as for instances in exclusive mode. Parallel Server uses global locks to coordinate data dictionary activity among multiple instances.

Parallel Cache Management uses several types of locks to control access to data and resources within a cluster. The way you configure the locks and the degree of granularity you use affect the performance of applications running on Oracle Parallel Server. The degree of granularity refers to the number of locks per instance.

You can most positively influence each type of locking by properly designing your applications. Second, setting initialization parameters and properly administering your cluster can also positively affect your system's overhead. On the other hand, improper lock use can cause your system to spend so much time synchronizing shared resources that you cannot achieve any speedup or scaleup.

Oracle Parallel Server uses two primary groups of locks as described in the following section:

- Local Locks
- Global Locks

Local Locks

There are two types of local locks, latches and enqueues. Latches are not Parallel Server-specific and are synchronized only within each instance. Latches thus do not have an effect on the global operations of clustered environments. Enqueues, however, can be both local to an instance and global to a cluster.

The following provides a brief description of these two types of locks:

- Latches
- Enqueues

Latches

Latches are simple, low level serialization mechanisms that protect in-memory data structures in the System Global Area. Latches do not protect data files, are automatic, and are held for a very short time in exclusive mode. As mentioned, because latches are synchronized within a node, they do not facilitate internode synchronization.

Enqueues

Enqueues are shared memory structures that serialize access to database resources. Enqueues are local to one instance if you do not enable Parallel Server. Or when you enable Parallel Server, enqueues can be global to a database. Enqueues are associated with a session or transaction and Oracle can use them in any of the following modes:

- Shared or "protected read"
- Exclusive
- Protected write
- Concurrent read
- Concurrent write
- Null

Enqueues are held longer than latches, have more granularity and more modes than latches, and protect more database resources. For example, if you request a table lock, or a DML lock, your request is assigned an "enqueue" lock. Enqueues are managed by the Distributed Lock Manager. When Parallel Server is enabled, most local enqueues become global enqueues.

Global Locks

Oracle Parallel Server synchronizes global locks among all active instances in a cluster. Global locks include two main types:

- Locks used by the Distributed Lock Manager for Parallel Cache Management
- Global locks, such as global enqueues, that Oracle synchronizes within a cluster to coordinate non-Parallel Cache Management resources

The Distributed Lock Manager tracks the status of all Oracle locking mechanisms. Oracle only creates global locks if you start an Oracle instance with Parallel Server enabled. The exception to this is the mount lock. The Distributed Lock Manager synchronizes global locks by communicating the status of a lock resource to all instances within an Oracle Parallel Server cluster.

Global locks are held by background processes within instances rather than by transactions. An instance owns a global lock that protects a resource, such as a data block or data dictionary entry, when the resource enters the instance's System Global Area. The Distributed Lock Manager manages locking only for resources accessed by more than one instance.

The following sections in this chapter are:

- Parallel Cache Management Coordination
- Non-Parallel Cache Management Coordination

Non-Parallel Cache Management Coordination

Non-parallel Cache Management includes coordination for resources other than data blocks. The Parallel Cache Management features of Oracle Parallel Server are described later in this chapter.

Non-Parallel Cache Management Locks

There are many different types of non-Parallel Cache Management locks. These control access to data files and control files. They also control library and dictionary caches, and perform various types of communication between instances. These locks do not protect data file blocks. Examples of these are Data Modification Language enqueues (table locks), transaction enqueues, and Data Definition Language locks or dictionary locks. The System Change Number (SCN), and the mount lock are global locks, not enqueues.

Note: The context of Oracle Parallel Server causes most local enqueues to become global; they can still be seen in the fixed tables and views that show enqueues, such as V$LOCK. The V$LOCK table does not, however, show global locks, such as SCN locks, mount locks, and Parallel Cache Management locks.

Non-PCM Global Locks

This section describes some of the most common non-PCM global locks. It covers the following information:

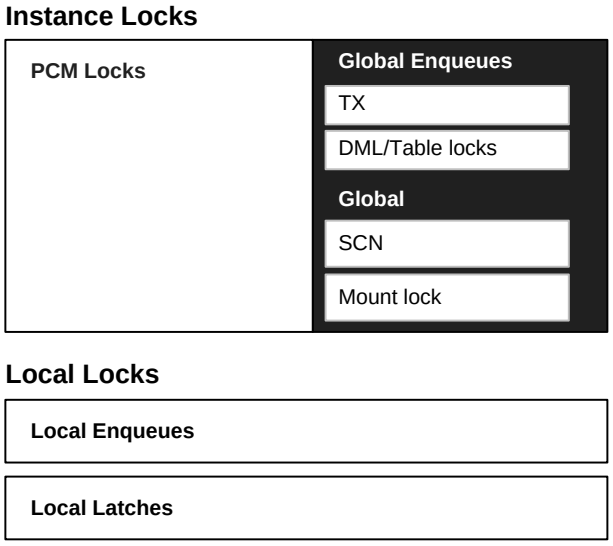
- Overview of Non-Parallel Cache Management Locks
- Transaction Locks
- Table Locks
- Library Cache Locks
- Dictionary Cache Locks
- Database Mount Lock

See Also: *Oracle8i Parallel Server Administration, Deployment, and Performance* for details on calculating the number of non-PCM resources and locks to configure in the Distributed Lock Manager.

Overview of Non-Parallel Cache Management Locks

This section explains how Oracle uses non-PCM locks to manage locks for transactions, tables, and other entities within an Oracle environment. Figure 4–1 highlights non-PCM locks in relation to other locks used in Oracle.

Figure 4–1 Oracle Locking Mechanisms: Non-PCM Locks



Whereas PCM locks are static (you allocate them when you design your application), non-PCM locks are very dynamic. Their number and corresponding space requirements will change as your system's initialization parameter values change.

See Also: *Oracle8i Reference* for descriptions of all non-PCM locks.

Transaction Locks

Row locks are locks that protect selected rows. A transaction acquires a global enqueue and an exclusive lock for each individual row modified by one of the following statements:

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

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

A transaction lock is acquired in exclusive mode when a transaction initiates its first change. It is held until the transaction does a COMMIT or ROLLBACK. SMON also acquires it in exclusive mode when recovering (undoing) a transaction. Transaction locks are used as a queuing mechanism for processes awaiting the release of an object locked by a transaction in progress.

Table Locks

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

Library Cache Locks

When a database object (table, view, procedure, function, package, package body, trigger, index, cluster, synonym) is referenced during parsing or compiling of a SQL (DML / DDL), PL / SQL, or Java statement, the process parsing or compiling the statement acquires the library cache lock in the correct mode. In Oracle8 the lock is held only until the parse or compilation completes (for the duration of the parse call).

Dictionary Cache Locks

The data dictionary cache contains information from the data dictionary, the meta-data store. This cache provides efficient access to the data dictionary.

Creating a new table, for example, causes the meta-data of that table to be cached in the data dictionary. If a table is dropped, the meta-data needs to be removed from

the data dictionary cache. To synchronize access to the data dictionary cache, latches are used in exclusive mode and in single shared mode. Global locks are used in multiple shared (parallel) mode.

In Oracle Parallel Server, the data dictionary cache on all nodes may contain the meta-data of a table that gets dropped on one instance. The meta-data for this table needs to be flushed from the data dictionary cache of every instance. This is performed and synchronized by global locks.

Database Mount Lock

The mount lock shows whether an instance has mounted a particular database. This lock is only used with Oracle Parallel Server. It is the only multi-instance lock used by Parallel Server in exclusive mode, where it prevents another instance from mounting the database in shared mode.

In Oracle Parallel Server single shared mode, this lock is held in shared mode. Another instance can successfully mount the same database in shared mode. In Parallel Server exclusive mode, however, another instance will not able to obtain the lock.

Parallel Cache Management Coordination

Understanding how Oracle Parallel Server synchronizes caches across instances can help you understand the overhead affecting system performance. Consider a five-node parallel server where a user drops a table on one of the nodes. Since each of the five dictionary caches has a copy of the definition of the dropped table, the node dropping the table from its cache must also cause the other four dictionary caches to drop their copies of the dropped table. Oracle Parallel Server handles this automatically through the Distributed Lock Manager. Users on the other nodes are notified of the change in lock status.

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

The processing within Oracle that requires synchronization includes:

- Block Level Locking
- Row Level Locking
- Space Management

- System Change Number

In Oracle Parallel Server exclusive mode, all synchronization is done within the instance. In shared mode, synchronization is accomplished with the help of the Distributed Lock Manager component that maintains the status of the global locks.

The most often required database resources are data blocks. Parallel Cache Management provides global locks to cover one or more types of data blocks. The types covered are:

- Data blocks
- Index blocks
- Undo blocks
- Segment headers

Parallel Cache Management locks ensure cache coherency by requiring that instances acquire a lock before modifying or reading a database block. Thus, Parallel Cache Management locks allow only one instance at a time to modify a block.

Parallel Cache Management of the buffer caches located on separate nodes provides Parallel Server cache coherency. The set of global constant (GC_*) initialization parameters associated with Parallel Cache Management buffer cache locks are not the same locks as those used with the dictionary cache, library cache, and so on.

Parallel Cache Management ensures that a master copy data block, also known as the "consistent read block" (the version of the block holding all the changes) is held in at least one of the System Global Areas in the cluster if it is to be changed. If an instance needs to read it, the current version of the block may reside in many buffer caches under shared locks. 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.

If a data block is modified in one buffer cache, then copies in other buffer caches are no longer current. New copies can be obtained after the modification operation completes.

Oracle only performs Parallel Cache Management lock operations for cache coherency when the current version of a data block is in one instance’s buffer cache and another instance requests that block for update.

Multiple transactions running on a single Oracle Parallel Server instance can share access to a set of data blocks for reading without additional global lock operations.

In this case, there is no contention or conflict. This remains true as long as the blocks are not needed for writing by transactions running on other instances.

Instances use global locks to indicate ownership of a resource master copy. When an instance becomes a database resource master or "owner", it also becomes owner of the global lock covering the resource with fixed locking. However, releasable locks are, of course, released.

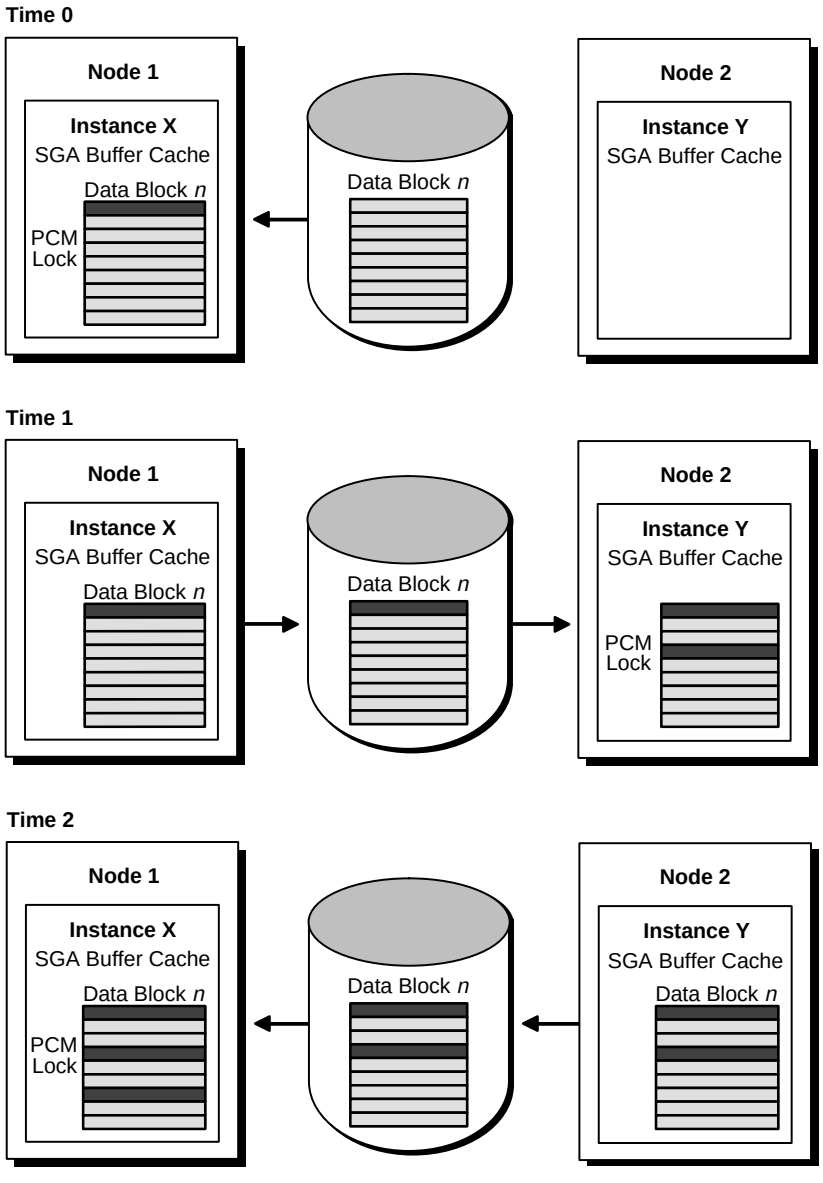
A master copy indicates it is an updatable copy of the resource. The instance only downgrades the global lock when another instance requests the resource for update. Once another instance owns the master copy of the resource, it becomes the owner of the global lock.

Example of Parallel Cache Management Processing

Consider the following example and the illustrations in Figure 4–2. This example assumes one Parallel Cache Management lock covers one block, although many blocks could be covered.

1. At "Time 0", Instance X becomes the owner of a Parallel Cache Management lock covering data block *n* containing row 1 and updates the row
2. Instance Y requests the block to update row 4
3. At "Time 1", Instance X writes the data block to disk and releases the Parallel Cache Management lock
4. Instance Y becomes the owner of the block and the Parallel Cache Management lock and then updates row 4
5. At "Time 2", Instance X requests the block to update row 7
6. Instance Y writes the data block to disk and releases the block and the Parallel Cache Management lock
7. Instance X becomes the owner of the block and Parallel Cache Management lock and updates row 7
8. Instance X commits its transaction and still owns the Parallel Cache Management lock and the master copy of the block until another instance requests the block

Figure 4–2 Multiple Instances Updating the Same Data Block



Parallel Cache Management Lock and Row Lock Independence

Parallel Cache Management locks and row locks operate independently. An instance can disown a Parallel Cache Management lock without affecting row locks held in the set of blocks covered by the Parallel Cache Management lock. A row lock is acquired during a transaction. A database resource such as a data block acquires a Parallel Cache Management lock when it is read for update by an instance. During a transaction, a Parallel Cache Management lock can therefore be disowned and owned many times if the blocks are needed by other instances.

In contrast, transactions do not release row locks until changes to the rows are either committed or rolled back. Oracle uses internal mechanisms for concurrency control to isolate transactions so modifications to data made by one transaction are not visible to other transactions until the transaction modifying the data commits. The row lock concurrency control mechanisms are independent of parallel cache management: concurrency control does not require Parallel Cache Management locks, and Parallel Cache Management lock operations do not depend on individual transactions committing or rolling back.

Global Lock Modes

An instance can acquire the global lock that covers a set of data blocks in either shared or exclusive mode, depending on the access type required.

- Exclusive lock mode allows the instance to update a set of blocks.

  If one instance needs to update a data block and a second instance already owns the global lock covering the block, the first instance uses the Distributed Lock Manager to request that the second instance disown the global lock, writing the block(s) to disk if necessary.

- Read lock mode only allows the instance to read blocks.

  Multiple instances can own a global lock in shared mode as long as they only intend to read, not modify, blocks covered by that global lock. Thus, all instances can be sure that their memory-resident copies of the block are current, or that they can read a current copy from disk without any global lock operations to request the block from another instance. This means instances do not have to disown global locks for the portion of a database accessed for read-only use, which may be a substantial portion of the time in many applications.

- Null lock mode allows instances to keep a lock without any permissions on the block(s).

This mode is used so that locks need not be continually obtained and released. Locks are simply converted from one mode to another.

Block Level Locking

Block access between instances is done on a per-block level. When an instance locks a block in exclusive mode, other instances cannot access the block. Every time Oracle tries to read a block from the database it must obtain a global lock. Ownership of the lock is thus assigned to the instance.

Since Oracle Parallel Server runs in environments with multiple memories, there can be multiple copies of the same data block in each instance's memory. Internode synchronization using the Distributed Lock Manager ensures the validity of all copies of the block: these block-level locks are the buffer cache locks.

Block level locking occurs only when Parallel Server is enabled. It is transparent to the user and to the application. Row level locking operates whether Parallel Server is enabled or disabled.

Comparing Parallel and Non-Parallel Cache Management Locks

Parallel Cache Management locks are typically more numerous than non-Parallel Cache Management locks. However, there are still enough non-Parallel Cache Management locks for which you must carefully plan adequate Distributed Lock Manager capacity. Typically 5% to 10% of locks are non-Parallel Cache Management. Non-Parallel Cache Management locks do not grow in volume the way Parallel Cache Management locks do.

You can control Parallel Cache Management locks in detail by setting initialization parameters to allocate the number desired. However, you have almost no control over non-Parallel Cache Management locks. You can attempt to eliminate the need for table locks by setting DML_LOCKS = 0 or by using the ALTER TABLE ENABLE / DISABLE TABLE LOCK command, but other non-Parallel Cache Management locks will still persist.

> **See Also:** *Oracle8i Parallel Server Administration, Deployment, and Performance* for information about allocating resources for Distributed Lock Manager locks.

5

Parallel Cache Management

This chapter explains how the Distributed Lock Manager in Oracle Parallel Server controls access to data. The Distributed Lock Manager also performs other synchronization tasks as described in Chapter 4. The overall process of managing data access and inter-instance coordination is known as "Parallel Cache Management." Topics in this chapter include:

- Parallel Cache Management and Lock Implementation
- Lock Duration and Granularity
- Coordination of Locking Mechanisms by the Distributed Lock Manager
- How Distributed Lock Manager Locks and Global Locks Relate
- Lock Elements and Parallel Cache Management Locks
- How Parallel Cache Management Locks Operate
- Number of Blocks Per Parallel Cache Management Lock
- How the DLM Grants and Coordinates Resource Lock Requests
- Specifying the Allocation and Duration of Locks

Parallel Cache Management and Lock Implementation

Oracle's Parallel Cache Management facility uses locks to coordinate shared data access by multiple instances. These locks are known as "Parallel Cache Management locks". Oracle Parallel Server also uses non-Parallel Cache Management locks to control access to data files and control files as explained in Chapter 4.

Parallel Cache Management locks are more numerous than non-Parallel Cache Management locks. They can also have a more volatile effect on performance because they control access to data blocks upon which the nodes in a cluster operate. For these reasons, it is critical that you accurately allocate Parallel Cache Management locks to ensure optimal performance.

Parallel Cache Management locks can cover one or more blocks of any class: data blocks, index blocks, undo blocks, segment headers, and so on. However, a given Parallel Cache Management lock can cover only one block class.

Parallel Cache Management ensures cache coherency by forcing requesting instances to acquire locks from instances that hold locks on data blocks before modifying or reading the blocks. Parallel Cache Management also allows only one instance at a time to modify a block. If a block is modified by an instance, the block must first be written to disk before another instance can acquire the Parallel Cache Management lock and modify it.

Parallel Cache Management locks use a minimum amount of communication to ensure cache coherency. The amount of cross-instance activity—and the corresponding performance of Oracle Parallel Server—is evaluated in terms of pings. A ping is when one instance requests a block that is held by another instance. To resolve this type of request, Oracle writes or "pings" the block to disk so the requesting instance can read it in its most current state.

Heavily loaded applications can experience significant locking activity, but they do not necessarily have excessive pinging. If data is well partitioned, the locking is local to each node—therefore pinging does not occur.

The Role of Cache Fusion in Resolving Cache Coherency Conflicts

Inter-instance references to data blocks and the resulting cache coherency issues are the main performance problems of Oracle Parallel Server. In most cases, proper partitioning resolves most contention problems.

In reality, however, most applications are not effectively partitioned, or are partitioned only to a limited extent. There are three types of cross-instance concurrent access:

- Reader / reader
- Reader / writer
- Writer / writer

Reader / reader concurrency occurs when two instances need to read the same data block. Oracle Parallel Server easily resolves this type of contention because multiple instances can share the same blocks for read access without cache coherency conflicts. The other types of contention, however, are more complex from a cache coherency point-of-view.

Reader / writer contention is in many cases the predominant form of concurrency in OLTP and hybrid applications. The ability to combine Decision Support System (DSS) and Online Transaction Processing (OLTP) in a typical application depends on Oracle Parallel Server's efficiency in resolving such conflicts.

In Oracle8*i*, concurrent write access to cached data blocks by multiple instances is managed by a disk-based "ping" protocol, in which current changes to a cached data block must be written to disk before another instance can read it and make changes to the block. The disk write must occur before exclusive access to a block is granted by the global locking mechanism.

In Oracle8*i*, Cache Fusion optimizes read / write concurrency by using the interconnect to directly transfer data blocks among instances. This eliminates I / O and reduces delays for block reads to the speed of the interprocess communication (IPC) and the interconnecting network. This also relaxes the strict requirements of data partitioning so that you can more efficiently deploy applications with mostly OLTP and mostly reporting modules.

Lock Duration and Granularity

Lock duration refers to the length of time for which a lock is associated with a resource. Lock granularity refers to the ratio of locks per data block.

Two Types of Lock Duration

Oracle Parallel Cache Management implements locks with two durations:

- Fixed Locks
- Releasable Locks

Fixed Locks

Fixed Parallel Cache Management locks are initially acquired in null mode. All specified fixed locks are allocated at instance startup, and de-allocated at instance shutdown. Because of this, fixed locks incur more overhead and longer startup times than releasable locks. The advantage of fixed Parallel Cache Management locks, however, is that they do not need to be continually acquired and released.

Fixed locks are pre-allocated and statically hashed to blocks at startup time. The first instance that starts up creates a Distributed Lock Manager resource and a Distributed Lock Manager lock, in null mode, on the resource for each fixed Parallel Cache Management lock. The instance then converts the mode of these locks to other modes as required. Each subsequent instance acquires a null mode lock at startup and then performs lock conversions as needed.

By default, fixed Parallel Cache Management locks are never released; each lock remains in the mode in which it was last requested. If the lock is required by another instance, it is converted to null mode. These locks are de-allocated only at instance shutdown.

Releasable Locks

Releasable locks are acquired and released as needed. This allows the instance to start up much faster than with fixed locks. A Distributed Lock Manager resource is created and an Distributed Lock Manager lock is obtained only when a user actually requests a block. Once a releasable lock has been created, it can be converted to various modes as required.

An instance can relinquish references to releasable lock resources during normal operations. The lock is released when it is required for reuse for a different block. This means that sometimes no instance holds a lock on a given resource.

Comparing Fixed and Releasable Locking With fixed duration locking, an instance never disowns a Parallel Cache Management lock unless another instance requests the lock. This minimizes the overhead of global lock operations in systems with relatively low contention for resources. With releasable locks, once the block is released, the lock on it is available for re-use. Non-Parallel Cache Management locks are disowned.

Releasable Parallel Cache Management locking is more dynamic than fixed locking. Releasable locks are allocated only as needed by the Distributed Lock Manager. At startup Oracle allocates lock elements that are obtained directly in the requested mode; this is normally shared or exclusive mode.

Two Forms of Lock Granularity

Oracle Parallel Cache Management implements two types of lock granularity:

- 1:1 Locks
- 1:n Locks

1:1 Locks

A 1:1 lock means one lock per block. This is the smallest granularity of locks and it is the default. 1:1 locks are useful when a database object is updated frequently by several instances. The advantages are:

- Conflicts occur only when the same block is needed by the two instances
- Only the required block is written to disk by the instance currently owning the Parallel Cache Management lock in exclusive mode

A disadvantage of 1:1 locking is that overhead is incurred for each block read, and performance is affected accordingly.

1:n Locks

1:n locks implies that a lock covers two or more data blocks as defined by the value for n. With 1:n locks, a few locks can cover many blocks and thus reduce lock operations. For read-only data, 1:n locks can perform faster than 1:1 locks during certain operations such as parallel execution.

If you partition data according to the nodes that are most likely to modify it, you can implement disjoint lock sets; each set belonging to a specific node. This can significantly reduce lock operations. 1:n locks are also beneficial if you have a large amount of data that a relatively small number of instances modify. If a lock is

already held by the instance that modifies the data, no lock activity is required for the operation.

> **See Also:** *Oracle8i Parallel Server Administration, Deployment, and Performance* for detailed information about configuring Parallel Cache Management locks.

The Cost of Locks

To effectively implement locks, carefully evaluate their relative costs. As a rule-of-thumb:

- Latches are inexpensive
- Local enqueues are more expensive
- Global locks and global enqueues are quite expensive

In general, global locks and global enqueues have an equivalent effect on performance. When Oracle Parallel Server is disabled, all enqueues are local. When Parallel Server is enabled, most enqueues are global. The length of time required to process these can vary by many milliseconds.

Microseconds, milliseconds, and tenths of seconds may seem negligible. However, imagine the cost of locks using grossly exaggerated values such as shown in the "Relative Time Required" column of Table 5–1.

Table 5–1 Comparing the Relative Cost of Locks

Class of Lock	Actual Time Required	Relative Time Required
Latches	1 microsecond	1 minute
Local Enqueues	1 millisecond	1000 minutes (16 hours)
Global locks (or Global Enqueues)	1/10 second	100,000 minutes (69 days)

Table 5–1 only shows relative examples to underscore the need to carefully calibrate lock use. In general, it especially critical to avoid unregulated global lock use.

> **See Also:** *Oracle8i Parallel Server Administration, Deployment, and Performance* for procedures to analyze the number of Parallel Cache Management locks applications use.

Coordination of Locking Mechanisms by the Distributed Lock Manager

The Distributed Lock Manager is a resource manager that is internal to the Oracle Parallel Server. This section explains how the Distributed Lock Manager coordinates locking mechanisms by covering the following topics:

- The Distributed Lock Manager Records Lock Information
- Lock Modes As Resource Access Rights
- Instances Map Database Resources to Distributed Lock Manager Resources
- How Distributed Lock Manager Locks and Global Locks Relate
- One Lock Per Instance on a Resource

Lock Modes As Resource Access Rights

Oracle may initially create a lock on a resource without granting access rights. If the instance later receives a request, Oracle converts the lock mode to obtain access rights. Figure 5–1 illustrates the levels of access rights or "lock modes" available through the Distributed Lock Manager. Table 5–2 lists and describes these lock modes.

Figure 5–1 Distributed Lock Manager Lock Modes: Levels of Access

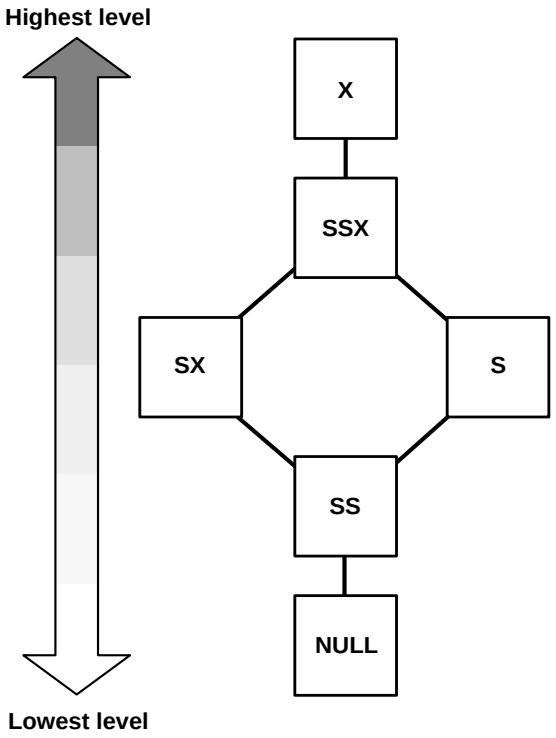


Table 5–2 Lock Mode Names

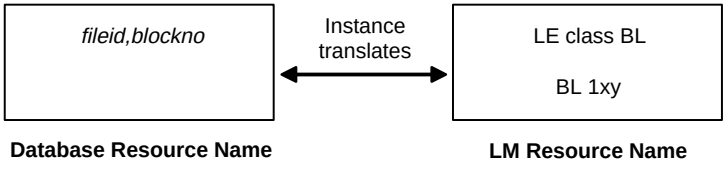
Lock Mode	Summary	Description
NULL	Null mode. No lock is on the resource.	Holding a lock at this level conveys no access rights. Typically, a lock is held at this level to indicate that a process is interested in a resource. Or it is used as a place holder. Once created, null locks ensure the requestor always has a lock on the resource; there is no need for the Distributed Lock Manager to constantly create and destroy locks when ongoing access is needed.
SS	Sub-shared mode (concurrent read). Read; there may be writers and other readers.	When a lock is held at this level, the associated resource can be read in an unprotected fashion: other processes can read and write the associated resource.
SX	Shared exclusive mode (concurrent write). Write; there may be other readers and writers.	When a lock is held at this level, the associated resource can be read or written in an unprotected fashion: other processes can both read and write the resource.
S	Shared mode (protected read). Read; no writers are allowed.	When a lock is held at this level, a process cannot write the associated resource. Multiple processes can read the resource. This is the traditional shared lock. In shared mode, any number of users can have simultaneous read access to the resource. Shared access is appropriate for read operations.
SSX	Sub-shared exclusive mode (protected write). One writer only; there may be readers	Only one process can hold a lock at this level. This allows a process to modify a resource without allowing other processes to simultaneously modify the resource at the same time. Other processes can perform unprotected reads. The traditional update lock.
X	Exclusive mode. Write; no other access is allowed	When a lock is held at this level, it grants the holding process exclusive access to the resource. Other processes cannot read or write the resource. This is the traditional exclusive lock.

Instances Map Database Resources to Distributed Lock Manager Resources

Each instance maps Oracle database resources to Distributed Lock Manager resources. For example, a 1:n lock on an Oracle database block with a given data block address, such as file 2 block 10, is translated as a "BL resource" with the class of the block and the lock element number, such as BL 9 1. The data block address is translated from the Oracle resource level to the Distributed Lock Manager resource level; the hashing function used is dependent on your GC_* parameter settings.

Note: For 1:1 locking, the database address is used as the second identifier, rather than the lock element number.

Figure 5–2 Resource Names and Distributed Lock Manager Resource Names



The Distributed Lock Manager Records Lock Information

The Distributed Lock Manager maintains an inventory of Oracle global locks and global enqueues held against system resources. It also acts as a negotiator when conflicting lock requests arise. In performing this function, the Distributed Lock Manager does not distinguish between Parallel Cache Management and non-Parallel Cache Management locks.

Sample Lock Manager Lock Mode and Resource Inventory

Figure 5–3 represents the Distributed Lock Manager as an inventory sheet that shows lock resources and the current status of the locks on those resources in an example Oracle Parallel Server environment.

Figure 5–3 Sample Distributed Lock Manager Resource and Lock Inventory

Integrated DLM

Resource	Locks
BL 100, 1	S
BL 101, 1	SSSNNN
BL 3000, 4	X
BL 3001, 4	SSS
BL 100, 6	NNN
BL 101, 6	X
BL 3000, 8	X
BL 3001, 8	N
BL 4000, 9	N

This inventory example includes all instances in this cluster. For example, resource BL 1, 101 is held by three instances with shared locks and three instances with null locks. Since Figure 5–3 shows up to six locks on one resource, at least six instances are running on this system.

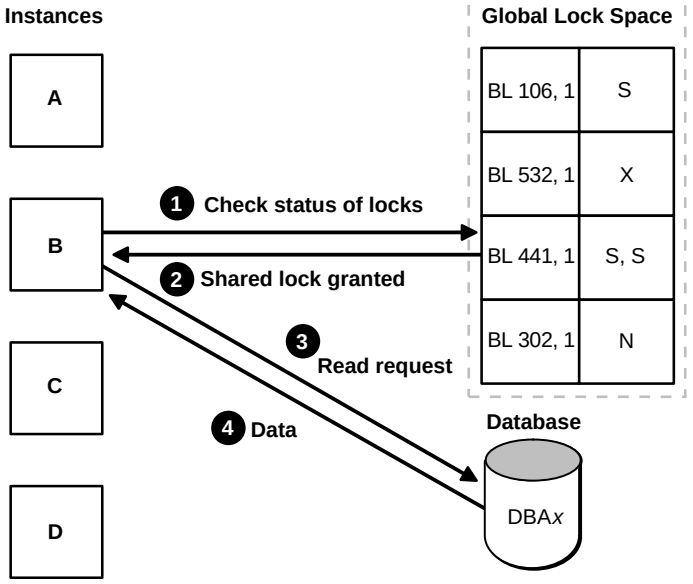
How Distributed Lock Manager Locks and Global Locks Relate

Figure 5–4 illustrates how Distributed Lock Manager locks and Parallel Cache Management locks relate. To allow instance B to read the value of data at data block address *x*, instance B must first check for locks on that address. The instance translates the block's database resource name to the Distributed Lock Manager resource name, and asks the Distributed Lock Manager for a shared lock to read the data.

As illustrated in Figure 5–4, the Distributed Lock Manager checks outstanding locks on the granted queue and determines there are already two shared locks on resource 441,BL1. Since shared locks are compatible with read-only requests, the Distributed Lock Manager grants a shared lock to instance B. The instance then

proceeds to query the database to read the data at data block address *x*. The database returns the data.

Figure 5–4 The Distributed Lock Manager Monitors the Status of Locks



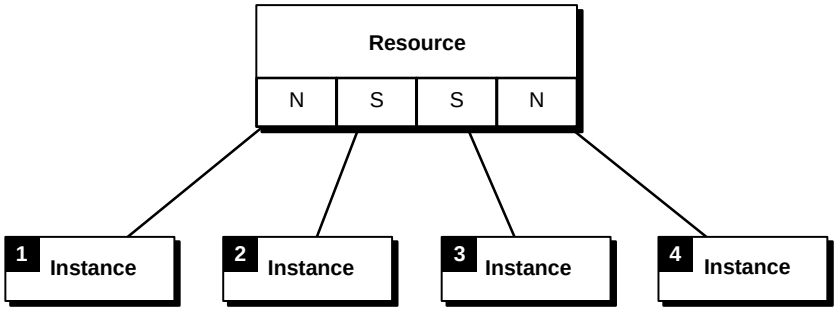
Note: The global lock space is cooperatively managed in a distributed fashion by the LMD processes of all instances.

If the required block already had an exclusive lock on it from another instance, then instance B would have to wait for this to be released. The Distributed Lock Manager would place the shared lock request from instance B on the convert queue. The Distributed Lock Manager would then notify the instance when the exclusive lock was removed and grant its request for a shared lock.

One Lock Per Instance on a Resource

Oracle uses only one lock per instance on any one Parallel Cache Management resource.The LCK0 process manages the assignment of this lock to the resource. As illustrated in Figure 5–5, if you have a four-instance system and require a buffer lock on a single resource, you actually need four locks—one per instance.

Figure 5–5 Resources Have One Lock Per Instance



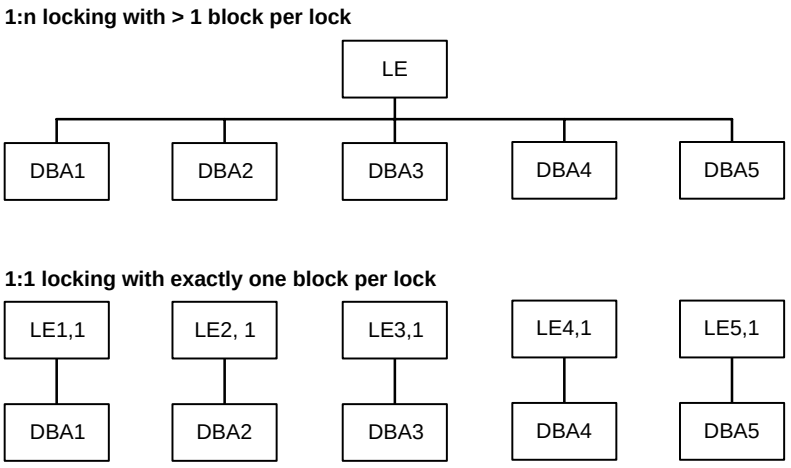
The number of locks on a non-Parallel Cache Management resource may depend on the type of resource, the application's behavior, and the configuration.

> **See Also:** Chapter 4 for more information on non-Parallel Cache Managements locks.

Lock Elements and Parallel Cache Management Locks

Figure 5–6 illustrates the correspondence of lock elements to blocks in fixed and releasable locking. A lock element (LE) is an Oracle-specific data structure representing a Parallel Cache Management lock. There is a one-to-one correspondence between a lock element and a Parallel Cache Management lock in the Distributed Lock Manager.

Figure 5–6 1:n Locking and 1:1 Locking



Lock Elements for Fixed Parallel Cache Management Locks

For both fixed Parallel Cache Management locks and releasable locks, you can specify more than 1 block per lock element. The difference is that by default, fixed Parallel Cache Management locks are not releasable: the lock element name is "fixed".

When the lock element is pinged due to a remote request, other modified blocks owned by that lock element are written along with the requested one. For example, in Figure 5–6, if LE is pinged when block DBA2 (Data Block Address 2) is needed, blocks DBA1, DBA3, DBA4, and DBA5 are all written to disk as well—if they have been modified.

Lock Elements for Releasable Parallel Cache Management Locks

With 1:1 locking, the name of the lock element is the name of the resource inside the Distributed Lock Manager. Although a fixed number of lock elements cover potentially millions of blocks, the lock element names change over and over as they are re-associated with blocks that are requested. The lock element name, for example, LE7,1, contains the database block address 7 and class 1 of the block it covers. Before a lock element can be reused, the lock must be released. You can then rename and reuse the lock element, creating a new resource in the Distributed Lock Manager if necessary.

When using 1:1 locking, you can configure your system with many more potential lock names, since they do not need to be held concurrently. However, the number of blocks mapped to each lock is configurable in the same way as 1:n locking.

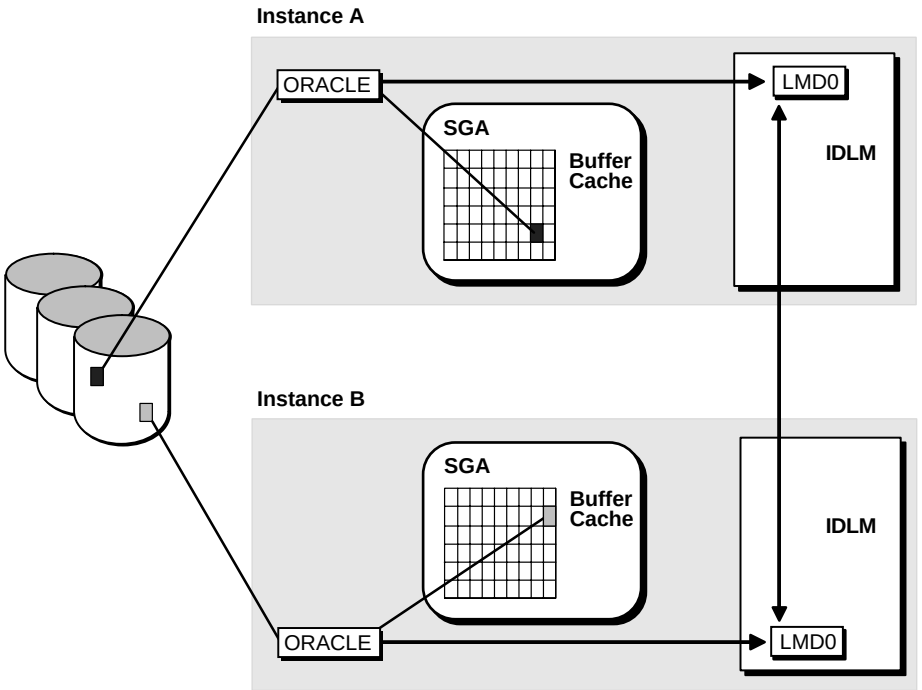
Lock Elements for 1:1 Parallel Cache Management Locks

In 1:1 locking you can set a one-to-one relationship between lock elements and blocks. Such an arrangement, illustrated in Figure 5–6, is called Data Block Address Locking. Thus if LE2,1 is pinged, only block DBA2 is written to disk.

How Parallel Cache Management Locks Operate

Figure 5–7 illustrates how Parallel Cache Management locks work. When instance A reads the black block for modification, it obtains the Parallel Cache Management lock for block. The same scenario occurs with the shaded block and Instance B. If instance B requires the black block, the block must be written to disk because instance A has modified it. The Oracle process communicates with the LMD processes to obtain the global lock from the Distributed Lock Manager.

Figure 5–7 How Parallel Cache Management Locks Work



Parallel Cache Management Locks Are Owned by Instance LCK Processes

Each instance has at least one LCK background process. If multiple LCK processes exist within the same instance, the Parallel Cache Management locks are divided among the LCK processes. This means that each LCK process is only responsible for a subset of the Parallel Cache Management locks.

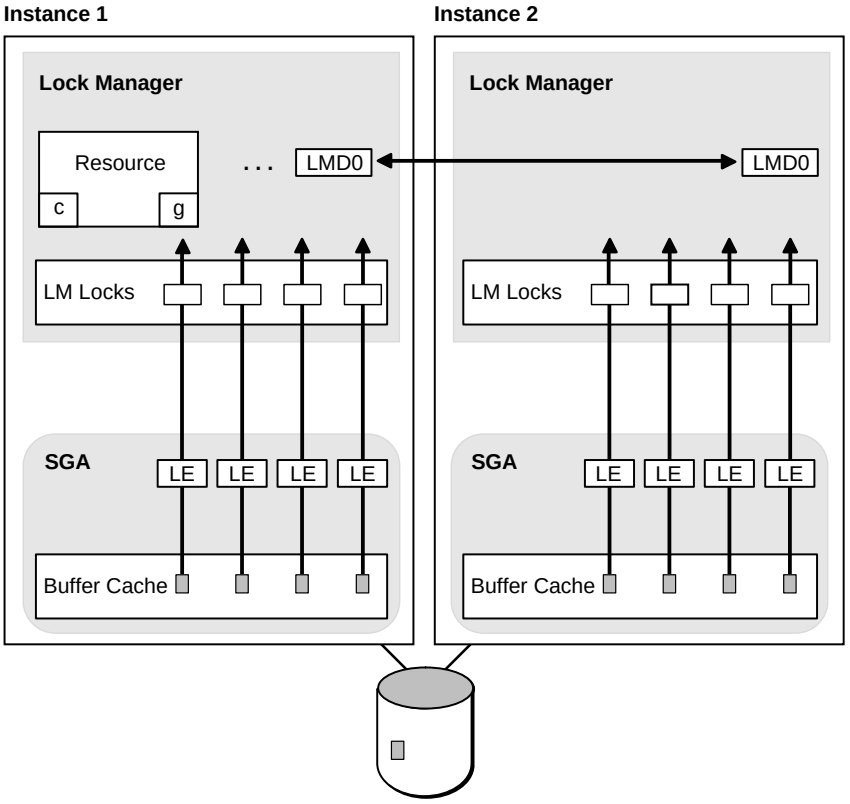
Multiple Instances Can Own the Same Locks

A Parallel Cache Management lock owned in shared mode is not disowned by an instance if another instance also requests the Parallel Cache Management lock in shared mode. Thus, two instances may have the same data block in their buffer caches because the copies are shared (no writes occur). Different data blocks covered by the same Parallel Cache Management lock can be contained in the buffer caches of separate instances. This can occur if all the different instances request the Parallel Cache Management lock in shared mode.

How 1:1 Locking Works

Figure 5–8 shows how 1:1 locking operates.

Figure 5–8 Lock Elements Coordinate Blocks (by 1:1 Locking)



The foreground process checks in the System Global Area to determine if the instance owns a lock on the block.

- If the lock is not owned or does not exist, the foreground process creates one, as either fixed or releasable, by releasing another lock
- If the instance owns the lock, but owns it in the incorrect mode, the foreground process converts the lock, for example, from shared to exclusive mode.

- Fixed locks, whether 1:n or 1:1, create lock element names in fixed mode, which is always valid. This mode is static so lock elements stay the same once they are allocated.
- Releasable locks, whether 1:n or 1:1, create a lock element in releasable mode; these lock element names can change, and the block or blocks covered can change. Lock elements in non-fixed mode can be valid, old, or free. If the valid bit is set then a lock is owned on the resource in the Distributed Lock Manager. If not set, there is no lock. If it is free, then there is a lock but we have unlinked the buffer from the lock element, so it is on the least recently used list of free lock elements.

Note: Valid lock elements have a lock in the Distributed Lock Manager; invalid lock elements do not. A free lock element indicates that a lock exists in the Distributed Lock Manager which is not currently linked to this buffer; it is waiting on the Least Recently Used list. If a lock element is old, then there is a valid lock handle for the old name. It must be given a new name before Oracle can use it.

Number of Blocks Per Parallel Cache Management Lock

The number of Parallel Cache Management locks assigned to data files and the number of data blocks in those data files determines the number of data blocks covered by a single Parallel Cache Management lock.

- If GC_FILES_TO_LOCKS is *not* set for a file, then releasable 1:1 locks are used with one lock for each block.
- If GC_FILES_TO_LOCKS *is* set for a file, then the number of blocks per Parallel Cache Management lock can be expressed as follows on a per file level. This example assumes values of GC_FILES_TO_LOCKS = 1:300,2:200,3-5:100.

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

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

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

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

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

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

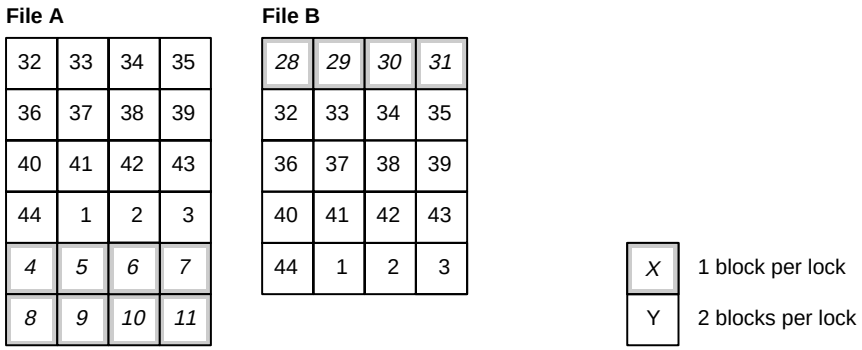
If you assign 1:n Parallel Cache Management locks to a set of data files collectively, then each lock usually covers one or more blocks in each file. Exceptions can occur when you specify contiguous blocks (using the "!*blocks*" option) or when a file contains fewer blocks than the number of locks assigned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

Example of Locks Covering Multiple Blocks

The following illustrates how 1:n Parallel Cache Management locks can cover multiple blocks in different files. Figure 5–9 assumes 44 Parallel Cache Management locks assigned to 2 files which have a total of 44 blocks. GC_FILES_TO_LOCKS is set to A,B:44

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

Figure 5–9 Fixed Parallel Cache Management Locks Covering Blocks in Multiple Files



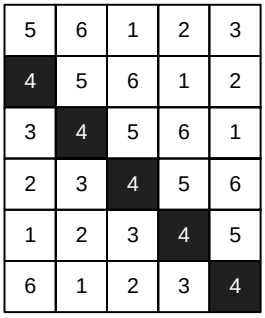
In Figure 5–9, locks 32 through 44 and 1 through 3 are used to cover 2 blocks each. Locks 4 through 11 and 28 through 31 cover 1 block each; and locks 12 through 27 cover no blocks at all!

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

Periodicity of Fixed Parallel Cache Management Locks

You should also consider the periodicity of Parallel Cache Management locks. Figure 5–10 shows a file of 30 blocks which is covered by 6 Parallel Cache Management locks. This file has 1:n locks set to begin with lock number 5. As suggested by the shaded blocks covered by Parallel Cache Management lock number 4, use of each lock forms a pattern over the blocks of the file.

Figure 5–10 Periodicity of Fixed Parallel Cache Management Locks



Pinging: Signaling the Need to Update

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

If the requesting instance wants the block for modification, then the holding instance's locks 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 is known as *pinging* a block.

The BSP (Block Server Process) uses the Distributed Lock Manager facility to signal a need between the two instances. If the requesting instance only wants the block in CR mode, the BSP of the holding instance transmits a CR version of the block to the requesting instance by way of the interconnect. In this scenario, pinging is much faster.

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

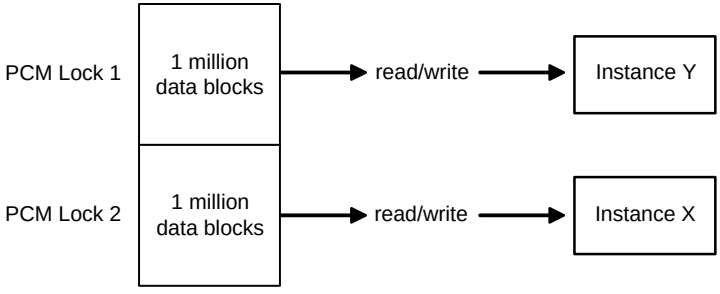
An instance can relinquish an exclusive lock on a block and still have a row lock on rows in it: pinging is independent of whether a commit has occurred. You can modify a block, but whether it is pinged is independent of whether you have made the commit.

Partitioning to Avoid Pinging

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

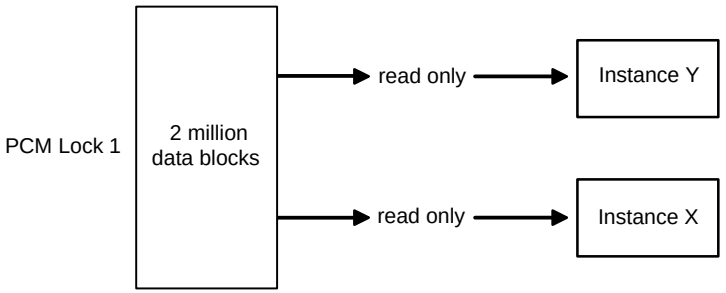
As shown in Figure 5–11, assume a single Parallel Cache Management lock covers one million data blocks in a table and the blocks in that table are read from or written into the System Global Area of instance X. Assume another single Parallel Cache Management lock covers another million data blocks in the table that are read or written into the System Global Area of instance Y. Regardless of the number of updates, there will be no forced reads or writes on data blocks between instance X and instance Y.

Figure 5–11 Partitioning Data to Avoid Pinging



With read-only data, both instance X and instance Y can hold the Parallel Cache Management lock in shared mode without causing pinging. This scenario is illustrated in Figure 5–12.

Figure 5–12 No Pinging of Read-Only Data



See Also: *Oracle8i Parallel Server Administration, Deployment, and Performance* for more information about partitioning applications to avoid pinging.

Lock Mode and Buffer State

The state of a block in the buffer cache relates directly to the mode of the lock held upon it. For example, if a buffer is in exclusive current (XCUR) state, you know that an instance owns the Parallel Cache Management lock in exclusive mode. There can be only one XCUR version of a block in the database, but there can be multiple SCUR versions. To perform a modification, a process must get the block in XCUR mode.

Finding the State of a Buffer

To see a buffer's state, check the STATUS column of the V$BH dynamic performance table. This table provides information about each buffer header.

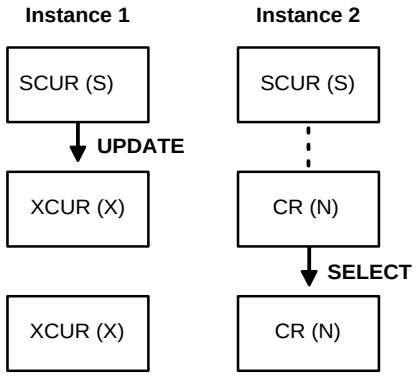
Table 5–3 Parallel Cache Management Lock Modes and Buffer States

Parallel Cache Management Lock Mode	Buffer State Name	Description
X	XCUR	Instance has an EXCLUSIVE lock for this buffer.
S	SCUR	Instance has a SHARED lock for this buffer.
N	CR	Instance has a NULL lock for this buffer.

How Buffer States and Lock Modes Change

Figure 5–13 shows how buffer state and lock mode change as instances perform various operations on a given buffer. Lock mode is shown in parentheses.

Figure 5–13 How Buffer States and Lock Modes Change



In Figure 5–13, the three instances start out with blocks in shared current mode, and shared locks. When Instance 1 performs an update on the block, its lock mode on the block changes to exclusive mode (X). The shared locks owned by Instance 2 and Instance 3 convert to null mode (N). Meanwhile, the block state in Instance 1 becomes XCUR, and in Instance 2 and Instance 3 it becomes CR. These lock modes are compatible.

Lock Modes May Be Compatible or Incompatible

When one process owns a lock in a given mode, another process requesting a lock in any particular mode succeeds or fails as shown in Table 5–4.

Table 5–4 Lock Mode Compatibility

Lock Requested: / Lock Owned	Null	SS	SX	S	SSX	X
NULL	SUCCEED	SUCCEED	SUCCEED	SUCCEED	SUCCEED	SUCCEED
SS	SUCCEED	SUCCEED	SUCCEED	SUCCEED	SUCCEED	FAIL
SX	SUCCEED	SUCCEED	SUCCEED	FAIL	FAIL	FAIL
S	SUCCEED	SUCCEED	FAIL	SUCCEED	FAIL	FAIL
SSX	SUCCEED	SUCCEED	FAIL	FAIL	FAIL	FAIL
X	SUCCEED	FAIL	FAIL	FAIL	FAIL	FAIL

How the DLM Grants and Coordinates Resource Lock Requests

This section explains how the Distributed Lock Manager coordinates resource lock requests by explaining the following topics:

- Lock Requests Are Queued
- Asynchronous Traps (ASTs) Communicate Lock Request Status
- Lock Requests Are Converted and Granted

The Distributed Lock Manager tracks all lock requests, granting requests for resources whenever permissible. Requests for resources that are not currently available are also tracked, and access rights are granted when these resources later become available. The Distributed Lock Manager inventories lock requests and communicates their statuses to users and to the internal processes involved in Parallel Cache Management.

Lock Requests Are Queued

The Distributed Lock Manager maintains two queues for lock requests:

Convert queue	If the Distributed Lock Manager cannot immediately grant a lock request, it is placed in the convert queue where waiting lock requests are tracked.
Granted queue	The Distributed Lock Manager tracks lock requests that have been granted in the granted queue.

Asynchronous Traps (ASTs) Communicate Lock Request Status

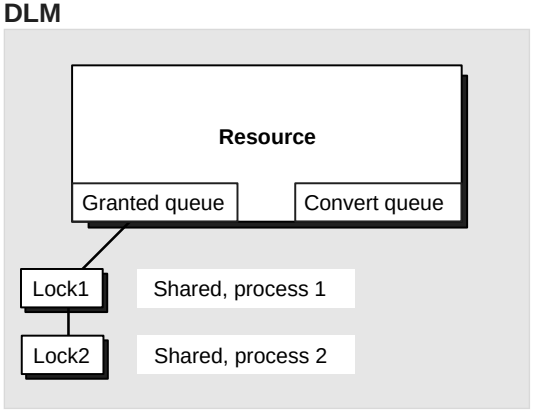
To communicate the status of lock requests, the Distributed Lock Manager uses two types of asynchronous traps (ASTs) or interrupts:

Acquisition AST	When the lock is obtained in the requested mode, an acquisition AST (a "wake up call") is sent to tell the requestor that the lock is granted.
Blocking AST	When a process requests a certain mode of lock on a resource, the Distributed Lock Manager sends a blocking AST to notify processes currently owning locks on that resource in incompatible modes. (Shared and exclusive modes, for example, are incompatible.) Upon notification, owners of locks can relinquish them to permit access by the requestor.

Lock Requests Are Converted and Granted

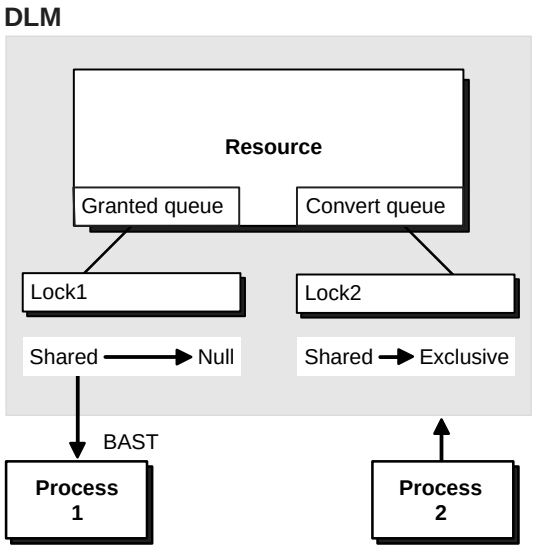
The following figures show how the Distributed Lock Manager handles lock requests. In Figure 5–14, shared lock request 1 has been granted on the resource to process 1, and shared lock request 2 has been granted to process 2. As mentioned, the Distributed Lock Manager tracks the locks in the granted queue. When a request for an exclusive lock is made by process 2, it must wait in the convert queue.

Figure 5–14 The Distributed Lock Manager Granted and Convert Queues



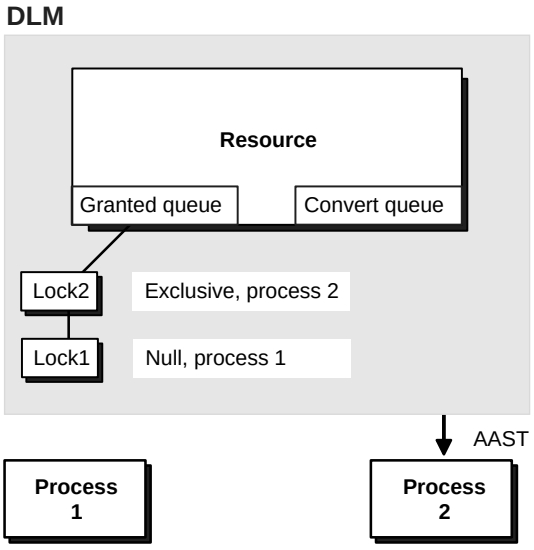
In Figure 5–15, the Distributed Lock Manager sends a blocking AST to Process 1, the owner of the shared lock, notifying it that a request for an exclusive lock is waiting. When the shared lock is relinquished by Process 1, it is converted to a null mode lock or released.

Figure 5–15 Blocking AST



An acquisition AST is then sent to alert Process 2, the requestor of the exclusive lock. The Distributed Lock Manager grants the exclusive lock and converts it to the granted queue. This is illustrated in Figure 5–16.

Figure 5–16 Acquisition AST



Specifying the Allocation and Duration of Locks

You allocate Parallel Cache Management locks to data files by specifying values for initialization parameters in parameter files that Oracle reads when starting up a database. For example, use the initialization parameter GC_FILES_TO_LOCKS to specify the number of Parallel Cache Management locks that cover the data blocks in a data file or set of data files.

Number of Blocks Per Parallel Cache Management Lock

This section explains the ways in which 1:n and 1:1 locks can differ in lock granularity.

1:N Locks for Multiple Blocks

You can specify lock-to-block ratios that protect a range of contiguous blocks within a file. Table 5–5 summarizes the situations in which 1:n locks are useful:

Table 5–5 When to Use 1:N Parallel Cache Management Locks

Situation	Reason
When the data is mostly read-only.	A few 1:n locks can cover many blocks without requiring frequent lock operations. These locks are released only when another instance needs to modify the data. 1:n locking can perform faster than 1:1 locking on read-only data with the Parallel Query Option. **I**f the data is strictly read-only, consider designating the tablespace itself as read-only. The tablespace will not then require any Parallel Cache Management locks.
When the data can be partitioned according to the instance which is likely to modify it.	1:n locks which are defined to match this partitioning allow instances to hold disjoint Distributed Lock Manager lock sets, reducing the need for Distributed Lock Manager operations.
When a large amount of data is modified by a relatively small set of instances.	1:n locks permit access to a new database block to proceed without Distributed Lock Manager activity, if the lock is already held by the requesting instance.

Using 1:n locks may cause extra cross-instance lock activity since conflicts may occur between instances that modify different database blocks. Resolution of false pinging may require writing several blocks from the cache of the instance that currently owns the lock. You can minimize or eliminate false pinging by correctly setting the GC_FILES_TO_LOCKS parameter.

1:1 Locking: Locks for One Block

If you create a one-to-one correspondence between Parallel Cache Management locks and data blocks, contention will occur only when instances need data from the same block. This level of 1:1 locking is also sometimes referred to as "DBA locking" where a "DBA" is the data block address of the data block. If you assign more than one block per lock, contention occurs as in 1:n locking.

On most systems, an instance could not possibly hold a lock for each block of a database since System Global Area memory or the Distributed Lock Manager capabilities would be exceeded. Therefore, instances acquire and release 1:1 locks as needed. Since 1:1 locks, lock elements, and resources are renamed in the Distributed Lock Manager and reused, a system can function properly with fewer of them. The value you set for the DB_BLOCK_BUFFERS parameter is the recommended minimum number of releasable locks you should allocate.

Selecting Lock Granularity

Use the information in Table 5–6 to best determine when to use either 1:n or 1:1 locks:

Table 5–6 When to To Use 1:n and 1:1 Locks

When to use 1:n locks...	When to use 1:1 locks...
■ Data is mostly read-only	■ Small amount of data is updated by many instances
■ Data can be partitioned	■ There is not enough memory for the configuration of 1:n locking
■ Large set of data is modified by a relatively small set of instances	

Simultaneously Using Fixed and Releasable Locking

Table 5–7 compares using both fixed and releasable locking at the same time.

Table 5–7 Using Fixed and Releasable Cache Management Locking

Fixed Parallel Cache Management Locks	Releasable Parallel Cache Management Locks
■ Allocated at instance startup, resulting in a slower startup ■ Released only at instance shutdown ■ Statically allocated to blocks at startup time, requiring more memory	■ Allocated when user requests a block, resulting in faster instance startup ■ Dynamically re-used by other blocks, requiring less memory

Group-Owned Locks

Group-based locking provides dynamic ownership: a single lock can be shared by two or more processes belonging to the same group. Processes in the same group can share and / or touch the lock without opening or converting a new and different lock. This is particularly important for the Multi-Threaded Server and XA.

Distributed Lock Manager Support for Multi-Threaded Server and XA

Oracle Parallel Server uses two forms of lock ownership:

Per-process ownership	Locks are commonly process-owned: that is, if one process owns a lock, then no other process can touch the lock.
Group-based ownership	With group-based locking, ownership becomes dynamic: a single lock can be used by two or more processes belonging to the same group. Processes in the same group can exchange and / or touch the lock without going to the Distributed Lock Manager grant and convert queues.

Group-based locking is an important Distributed Lock Manager feature for Oracle Multi-Threaded Server (MTS) and XA library functionality.

MTS	Group-based locking is used for Oracle MTS configurations. Without it, sessions could not migrate between shared server processes. In addition, load balancing may be affected, especially with long running transactions.
XA libraries	With Oracle XA libraries, multiple sessions or processes can work on the transaction; they therefore need to exchange the same locks, even in exclusive mode. With group-based lock ownership, processes can exchange access to an exclusive resource.

Memory Requirements for the Distributed Lock Manager

The user-level Distributed Lock Manager can normally allocate as many resources as you request; your process size, however, will increase accordingly. This is because you are mapping the shared memory where locks and resources reside into your address space. Thus, the process address space can become very large.

Make sure that the Distributed Lock Manager is configured to support all resources your application requires.

Part III

Implementing Oracle Parallel Server

Part III describes several topics specific to Oracle Parallel Server implementation. The chapters in this part are:

- Chapter 6, "Oracle Parallel Server Components"
- Chapter 7, "Oracle Parallel Server Storage Considerations"
- Chapter 8, "Scalability and Oracle Parallel Server"
- Chapter 9, "High Availability and Oracle Parallel Server"

6

Oracle Parallel Server Components

This chapter describes components for implementing Oracle Parallel Server. The topics in this chapter are:

- Instance and Database Components for Oracle Parallel Server
- Cache Fusion Processing and the Block Server Process
- System Change Number Processing

Instance and Database Components for Oracle Parallel Server

This chapter discusses the Oracle Parallel Server-specific concepts of implementation from a process and component point-of-view. Each Oracle Parallel Server instance has its own System Global Area and background processes common to single-instances. Each instance also has its own set of control files, data files, and redo logs. Instances within a cluster also share or have access to all System Global Areas, control and data files, and redo logs throughout the cluster.

Other Parallel Server-specific components described in the following sections. These components enable parallel processing and facilitate internode communication.

Parallel Server-Specific Processes

In addition to the processes common to single-instances, such as PMON, SMON, DBWR, and so on. Oracle Parallel Server instances have additional processes to facilitate resource sharing among nodes in a cluster.

> **See Also:** *Oracle8i Concepts* for more information on Oracle processes.

Four Parallel Server-specific processes facilitate resource sharing. The first three processes work with the Distributed Lock Manager component to manage global locks and resources:

- LMON – The "Lock Monitor Process" monitors the entire cluster to manage global locks and resources. LMON manages instance and process deaths and the associated recovery for the Distributed Lock Manager. In particular, LMON handles the part of recovery associated with global locks.
- LMD – The "Lock Manager Daemon" is the lock agent process that manages lock manager service requests for Parallel Cache Management locks to control access to global locks and resources. The LMDN process also handles deadlock detection and remote lock requests. Remote lock requests are requests originating from another instance.
- LCK – The "Lock Process" manages non-Parallel Cache Management locking to coordinate shared resources and remote requests for those resources.
- BSP – The "Block Server Process" rolls back uncommitted transactions for blocks requested by cross-instance read / write requests and sends the consistent read block to the requestor.

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

When an instance fails while in shared mode, another instance's SMON detects the failure and recovers the failed instance. The LMON process of the instance performing recovery re-masters outstanding PCM locks for the failed instance.

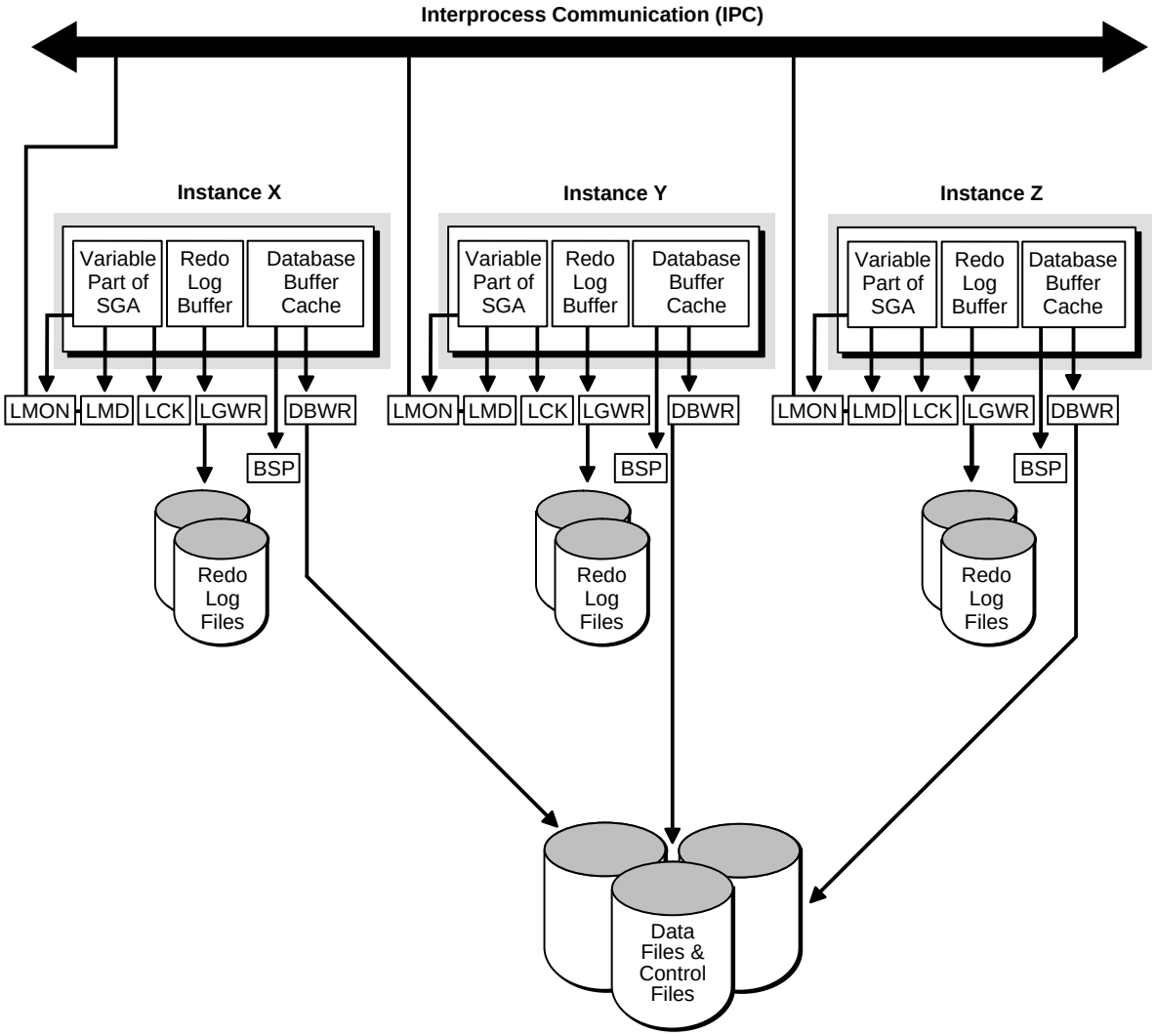
Foreground Processes and Foreground Lock Acquisition

Foreground processes are the processes associated with user sessions. One instance communicates lock requests directly to remote LMD processes in other instances. Foreground processes send request information such as the resource name it is requesting a lock for and the mode in which it needs the lock. The Distributed Lock Manager processes the request asynchronously, so the foreground process waits for the request to complete before closing the request.

Overview of Oracle Parallel Server Processes

The Oracle Parallel Server-specific processes and some of the processes that Oracle also uses in single-instance environments appear in Figure 6–1.

Figure 6–1 Basic Elements of Oracle Parallel Server



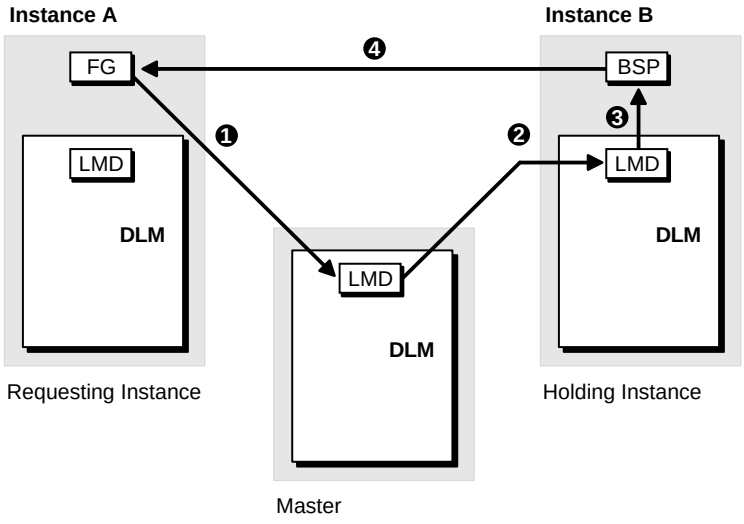
Cache Fusion Processing and the Block Server Process

Cache Fusion resolves cache coherency conflicts when one instance requests a block held in exclusive mode by another instance. In such cases, Oracle transfers a consistent-read version of the block directly from the memory cache of the holding instance to the requesting instance. Oracle does this without writing the block to disk.

As mentioned, Cache Fusion uses the Block Server Process to roll back uncommitted transactions. BSP then sends the consistent read block directly to the requestor. The state of the block is consistent as of the point in time at which the request was made by the requesting instance. Figure 6–2 illustrates this process.

Cache Fusion does this only for consistent read, reader / writer requests. This greatly reduces the number of lock downgrades and the volume of inter-instance communication. It also increases the scalability of certain applications that previously were not likely Oracle Parallel Server candidates, such as OLTP and hybrid applications.

Figure 6–2 Step-by-step Cache Fusion Processing



9. The requestor's FG (foreground) process sends a lock request message to the master node. The requesting node, the holding node, or an entirely separate node can serve as the master node.

10. The LMD process of the master node forwards the lock request to the LMD process of the holding node that has an exclusive lock on the requested block.
11. The holding node's LMD process handles the in-coming message and requests the holding instance's BSP to prepare a consistent read copy of the requested block.
12. BSP prepares and sends the requested block to the requestor's FG process.

System Change Number Processing

The System Change Number (SCN) is a logical timestamp that Oracle uses to order events within a single instance and across all instances. For example, Oracle assigns an SCN to each transaction. Conceptually, there is a global serial point that generates SCNs. In practice, however, SCNs can be read and generated in parallel. One of the SCN generation schemes is called the Lamport SCN generation scheme.

In single-instance Oracle, the System Global Area maintains and increments SCNs from an instance that has mounted the database in exclusive mode. In Oracle Parallel Server shared mode, the SCN must be maintained globally. Its implementation may vary from platform to platform. The SCN may be handled by the Distributed Lock Manager, by the Lamport SCN scheme, or by using a hardware clock or dedicated SCN server.

How Lamport SCN Generation Works

The Lamport SCN generation scheme is fast and scalable because it generates SCNs in parallel on all instances. In this scheme, all messages across instances, including lock messages, piggyback SCNs. Piggybacked SCNs propagate causalities within Oracle. As long as causalities are respected in this way, multiple instances can generate SCNs in parallel, with no need for extra communication among these instances.

On most platforms, Oracle uses the Lamport SCN generation scheme when the MAX_COMMIT_PROPAGATION_DELAY is larger than a platform-specific threshold. This is generally the default. This value is typically set to seven seconds. You can examine the alert log after instance startup to see whether the Lamport SCN generation scheme is in use. If this value is smaller than seven, Oracle uses the lock SCN generation scheme?

7

Oracle Parallel Server Storage Considerations

This chapter describes storage considerations for Oracle Parallel Server applications. Topics in this chapter include:

- Oracle Parallel Server-Specific Storage Issues
- Space Management and Free List Groups
- Controlling Extent Allocation

Oracle Parallel Server-Specific Storage Issues

This section describes storage issues specific to Oracle Parallel Server. This section discusses:

- Data Files
- Redo Log Files
- Rollback Segments

Data Files

All Oracle Parallel Server instances access the same data files. The composition of database files is the same for Oracle in parallel mode and in exclusive mode. You do not have to alter the data files to start Oracle in parallel mode.

> **Note :** Oracle Parallel Server requires special naming conventions for data files as specified in *Oracle8i Parallel Server Setup and Configuration Guide.*

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

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

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

If verification fails for any instance, diagnose and fix the problem, then use the ALTER SYSTEM CHECK DATAFILES statement to verify access. This statement has a GLOBAL option, which is the default, that makes all instances verify access to online data files. It also has a LOCAL option that makes the current instance verify access.

ALTER SYSTEM CHECK DATAFILES makes online data files available to the instance or instances for which access is verified.

Oracle cannot recover from instance failure or media failure unless the instance that performs recovery can verify access to all required online data files.

Oracle automatically maps absolute file numbers to relative file numbers. Use of Oracle Parallel Server does not affect these values. Query the V$DATAFILE view to see both numbers for your data files.

Redo Log Files

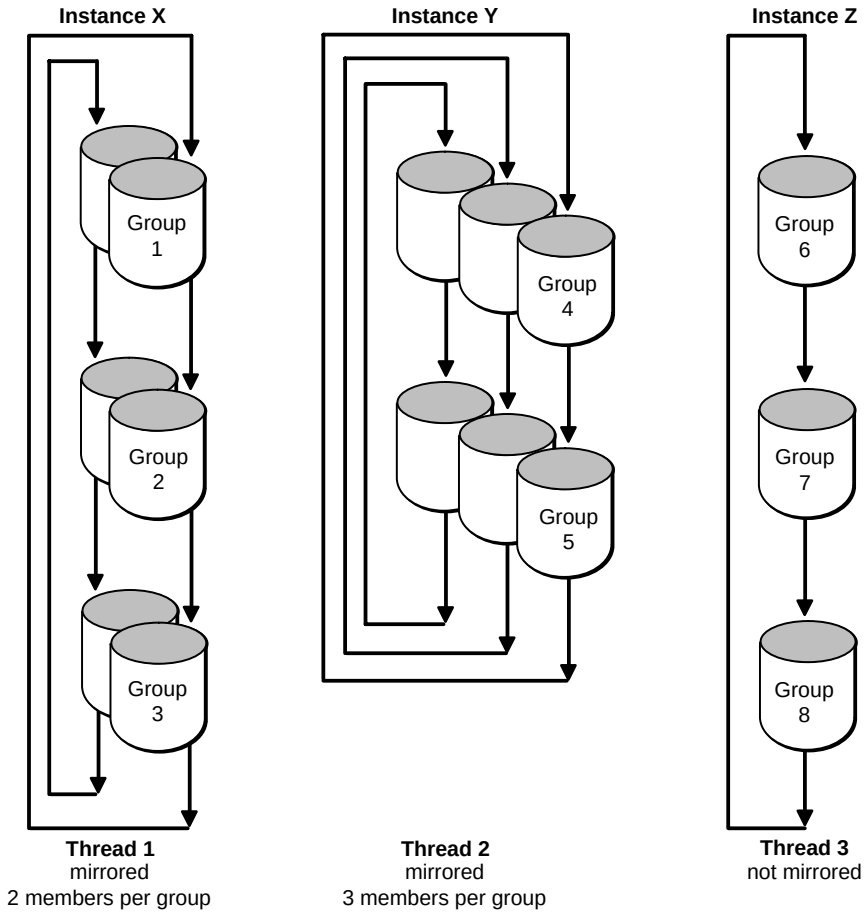
In Oracle Parallel Server, each instance writes to its own set of online redo log files. The redo written by a single instance is called a "thread of redo". Each online redo log file is associated with a particular thread number. When an online redo log is archived, Oracle records its thread number to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

A "private thread" is a redo log created using the ALTER DATABASE ADD LOGFILE command with the THREAD clause. A "public thread" is a redo log created using the ALTER DATABASE ADD LOGFILE but without specifying a THREAD clause.

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

Online redo log files can be multiplexed, or "mirrored". A multiplexed redo log consists of two or more groups of files and all members of a group are written to concurrently when that group is active. Figure 7–1 shows the threads of redo for three instances of Oracle Parallel Server.

Figure 7–1 Threads of Redo



- Instance X uses thread 1, which contains three *groups* of online redo log files, groups 1, 2, and 3. Thread 1 is multiplexed, that is, each group has two copies, or *members*, of the redo log file.
- Instance Y uses thread 2, which contains two groups of online redo log files, groups 4 and 5. Thread 2 is multiplexed, with three members per group.
- Instance Z uses thread 3, which contains three groups of online redo log files, groups 6, 7, and 8, that are not multiplexed.

Note: The Database Configuration Assistant creates two log files for each group and one group per instance.

Group numbers must be unique within the database, therefore they are unique within a thread. However, the order of assigning groups to threads, and threads to instances, is arbitrary.

For example, although in Figure 7–1 thread 1 contains groups 1, 2, and 3 while thread 2 contains groups 4 and 5, you could instead assign groups 2, 4, and 5 to thread 1 while assigning groups 1 and 3 to thread 2. The V$LOGFILE view displays the group number associated with each redo log file.

Although it is possible to have different numbers of groups and members per thread, we recommend that all threads be configured to a common standard to facilitate administration.

Different instances of Oracle Parallel Server can have different degrees of mirroring, or different numbers of members per group. The different instances can also have different numbers of groups. For example, one instance could have three groups with two members per group, a second instance could have four non-multiplexed log files, and a third instance could have two groups with four members per group. While such a configuration may be inconvenient to administer, it may be necessary to achieve the full potential of the system.

Each instance must have at least two groups of online redo log files. When the current group fills, an instance begins writing to the next log file group. At a log switch, information is written to the control file that can be used to identify the filled group and its thread number after it has been archived.

The number of redo log files about which the control file can keep information is limited by the value of the MAXLOGHISTORY option of the CREATE DATABASE statement. Only one member per group is needed. In Oracle Parallel Server, set the value of MAXLOGHISTORY higher than you normally would in single instance Oracle. This is because in Oracle Parallel Server, the history of multiple redo log files must be tracked.

Note: MAXLOGHISTORY is useful for sites with very demanding availability requirements. This option can help you administer recovery, especially when there are many instances and many log files.

See Also: *Oracle8i Concepts* for a full description of multiplexed redo log files.

Rollback Segments

This section describes rollback segments as they relate to Oracle Parallel Server.

Rollback Segments in Oracle Parallel Server

Rollback segments contain information that Oracle requires to maintain read consistency and to be able to undo changes made by transactions that roll back or abort. Each instance in Oracle Parallel Server shares use of the SYSTEM rollback segment and requires at least one dedicated rollback segment per instance.

Both private and public rollback segments can be acquired at instance startup and used exclusively by the acquiring instance until taken offline or when the acquiring instance is shut down as specified in the rollback segment parameter. Private rollback segments are unique to a particular instance; other instances cannot use them. A public rollback segment is offline and not used by any instance until an instance that needs an extra rollback segment starts up, acquires it, and brings it online. Once online, the acquiring instance uses the public rollback segment exclusively.

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

Oracle Parallel Server needs at least as many rollback segments as the maximum number of concurrent instances plus one; the extra one is for the SYSTEM rollback segment. An instance cannot start up shared without exclusive access to at least one rollback segment, whether it is public or private.

You can create new rollback segments in any tablespace. To reduce contention between rollback data and table data, partition your rollback segments in a separate tablespace. This also facilitates taking tablespaces offline because a tablespace cannot be taken offline if it contains active rollback segments.

In general, make all rollback segment extents the same size by specifying identical values for the storage parameters INITIAL and NEXT.

The data dictionary view DBA_ROLLBACK_SEGS shows each rollback segment's name, segment ID number, and owner (PUBLIC or other).

> **See Also:** *Oracle8i Administrator's Guide* for information about contention for a rollback segment and for information on the performance implications of adding rollback segments.

Parameters Controlling Rollback Segments

These initialization parameters control rollback segment use:

ROLLBACK_SEGMENTS	specifies the names of rollback segments that the instance acquires at startup.
GC_ROLLBACK_LOCKS	reserves instance locks to reduce contention for blocks containing rollback entries. In particular, it reserves instance locks for deferred rollback segments, that contain rollback entries for transactions in tablespaces that were taken offline.

> **See Also:** *Oracle8i Concepts* for more information about data blocks, extents, and segments.

Public and Private Rollback Segments

Public and private rollback segments do not have performance differences. However, private rollback segments provide more control over the matching of instances with rollback segments. This allows you to locate the rollback segments for different instances on different disks to improve performance. Therefore, use private rollback segments to reduce disk contention in high-performance systems.

Public rollback segments form a pool of rollback segments that can be acquired by any instance needing an additional rollback segment. Using public rollback segments can be disadvantageous, however, when instances are shut down and started up at the same time. For example, instance X shuts down and releases public rollback segments. Instance Y starts up and acquires the released rollback segments. Finally, instance X starts up and cannot acquire its original rollback segments. Acquiring a public rollback segment can also be made at startup if TRANSACTIONS and TRANSACTIONS_PER_ROLLBACK_SEGMENTS are not properly set.

You can use public rollback segments to improve space utilization. If you create only one large public rollback segment for long-running transactions that run on

different instances each month, the rollback segment can be taken offline and brought back online or "moved" from one instance to another to better serve instances with the heavier workloads.

By default a rollback segment is private and is used by the instance specifying it in the parameter file. Specify private rollback segments using the parameter ROLLBACK_SEGMENTS.

Once a public rollback segment is acquired by an instance, it is then used exclusively by that instance.

Once created, both public and private rollback segments can be brought online using the ALTER ROLLBACK SEGMENT command.

Note: An instance needs at least one rollback segment or it will not be able to start.

How Instances Acquire Rollback Segments

When an instance starts, it uses the TRANSACTIONS and TRANSACTIONS_PER_ROLLBACK_SEGMENT initialization parameters to determine how many rollback segments to acquire as shown in the following equation:

$$\frac{TRANSACTIONS}{TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT} = total\_rollback\_segments\_required$$

The value for *total_rollback_segments_required* is rounded up.

At startup, an instance attempts to acquire rollback segments by executing the following steps:

- The instance first acquires any private rollback segments specified by the ROLLBACK_SEGMENTS initialization parameter. If the *total_private_rollback_segments* number is more than the *total_rollback_segments_required*, then no further action is taken to acquire rollback segments.
- If the initialization file does not specify private rollback segments, the instance attempts to acquire *public* rollback segments.

- If the *total_private_rollback_segments* falls short of the *total_rollback_segments_required*, then the instance attempts to make up the difference by acquiring public rollback segments.
- If only one private rollback segment is specified and acquired, or one public rollback segment is acquired, the instance starts up, even if one rollback segment is below the *total_rollback_segments_ required*. In this case, Oracle generates a message.
- If a private rollback segment cannot be brought online at instance startup, the startup fails and Oracle generates a message.

Space Management and Free List Groups

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

- How Oracle Handles Free Space
- Free Lists and Free List Groups
- Partitioning Data with Free List Groups
- Associating Instances, Users, and Locks with Free List Groups
- SQL Options for Managing Free Space

How Oracle Handles Free Space

Oracle Parallel Server enables transactions running on separate instances to insert and update data in the same table concurrently. This occurs without contention to locate free space for new records. However, to take advantage of this capability, you must accurately manage free space in your database using several structures that are described in this section.

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

When Oracle allocates new extents to a table, Oracle adds their blocks to the master free list. This can eventually result in contention for free space among multiple instances on Oracle Parallel Server because the free space contained in newly allocated extents cannot be reallocated to any group of free lists. You can have more control over free space if you specifically allocate extents to instances; in this way you minimize free space contention.

Segments, Extents, and the High Water Mark

A segment is a unit of logical database storage. Oracle allocates space for segments in smaller units called extents. An extent is a specific number of contiguous data blocks allocated for storing a specific type of information. A segment thus comprises a set of extents allocated for a specific type of data structure. For example, each table's data is stored in its own data segment, while each index's data is stored in its own index segment.

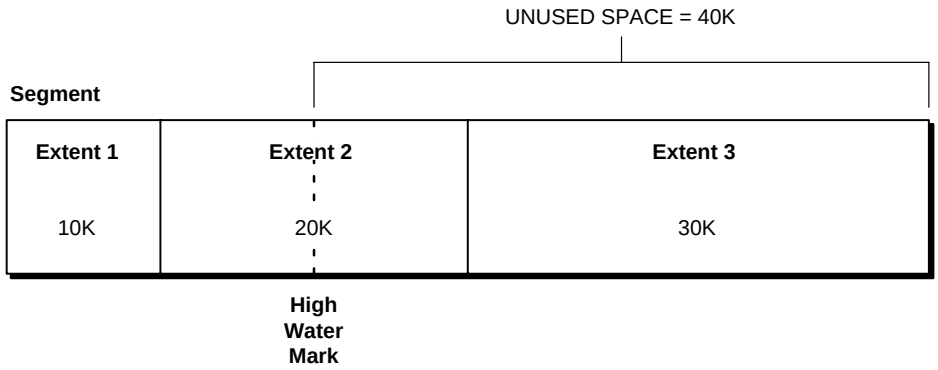
A segment's extents are stored in the same tablespace. However, they may or may not be contiguous on disk. The segments can span files, but individual extents cannot.

Although you can allocate additional extents, the blocks themselves are allocated separately. If you allocate an extent to a specific instance, the blocks are immediately allocated to the free list. However, if the extent is not allocated to a specific instance, then the blocks themselves are allocated only when the high water mark moves.

The high water mark is the boundary between used and unused space in a segment. As requests for new free blocks that cannot be satisfied by existing free lists are received, the block to which the high water mark points becomes a used block, and the high water mark is advanced to the next block. In other words, the segment space to the left of the high water mark is used, and the space to the right of it is unused.

Figure 7–2 shows a segment consisting of three extents containing 10K, 20K, and 30K of space, respectively. The high water mark is in the middle of the second extent. Thus, the segment contains 20K of used space to the left of the high water mark, and 40K of unused space to the right of the high water mark.

Figure 7–2 High Water Mark



See Also: *Oracle8i Concepts* for more information about segments and extents.

Free Lists and Free List Groups

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

Blocks in free lists contain free space greater than PCTFREE. This is the percentage of a block to be reserved for updates to existing rows. In general, blocks included in process free lists for a database object must satisfy the PCTFREE and PCTUSED constraints.

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

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

- Master free lists–a list of blocks containing available space drawn from any extent in a table
- Transaction free lists–a list of blocks freed by uncommitted transactions

Because Parallel Server has multiple instances, free lists alone cannot solve contention problems. Free list groups, however, effectively reduce pinging between instances.

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

Free List Groups

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

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

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

Note: In Oracle Parallel Server, always use free list groups and free lists.

Avoiding Contention for Segment Header and Free Lists

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

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

In multi-instance environments, as illustrated in Figure 7–3, free lists provide free data blocks from available extents to different instances. You can partition multiple free lists so that extents are allocated to specific database instances. Each instance hashes to one or more free list groups, and each group's header block points to free lists. Without free list groups, every instance must read the segment header block to access the free lists.

Figure 7–3 Contention for the Segment Header

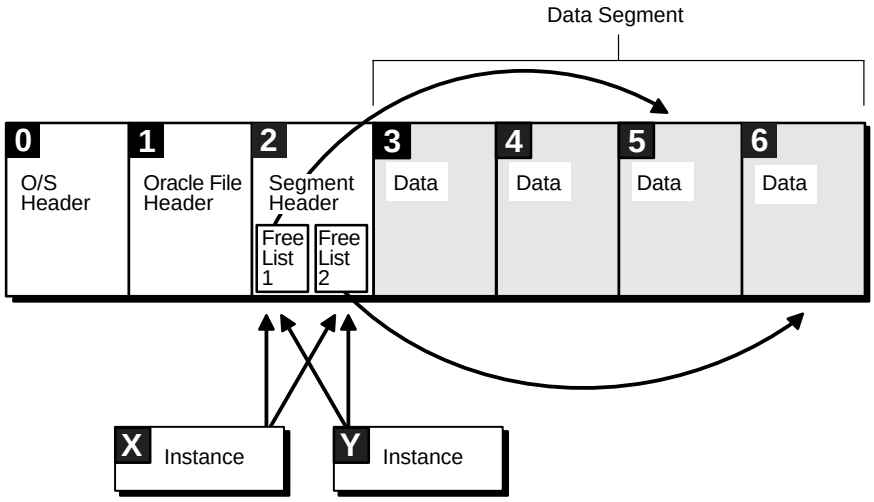
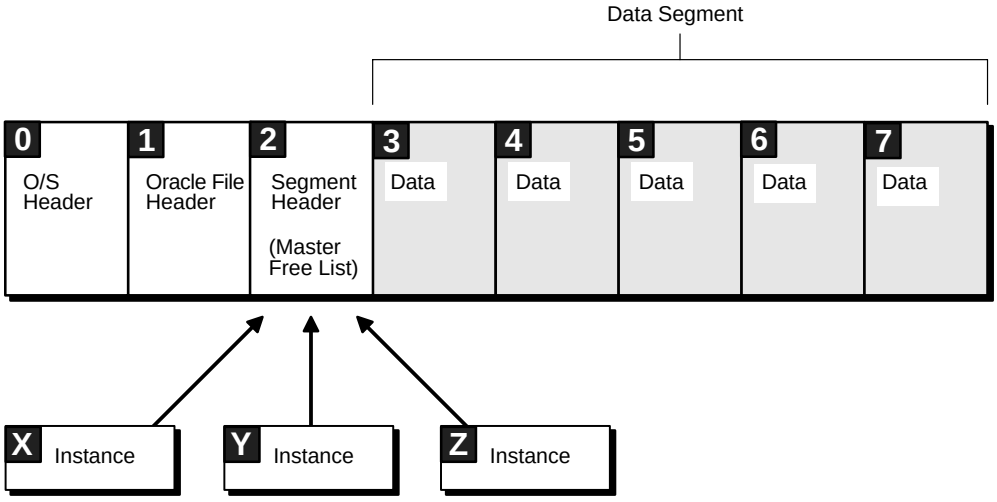


Figure 7–4 shows the blocks of a file in which the master free list is stored in the segment header block. Three instances are forced to read this block to obtain free space. Because there is only one free list, there is only one insertion point. Free list groups help reduce contention by spreading this insertion point over multiple blocks, each of which will be accessed less frequently.

Figure 7–4 Contention for the Master Free List



Locally Managed Tablespaces

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

Free List Group Examples

This section describes two free list group examples:

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

Basic Free List Group Example

Figure 7–5 illustrates the division of free space for a table into a master free list and two free list groups, each of which contains three free lists. This example involves a well-partitioned application in which deletes occur. The master free list pictured is the master free list for this particular free list group.

The table was created with one initial extent, after which extents 2 and 5 were allocated to instance X, extents 3 and 4 were allocated to instance Y, and extent 6 was allocated automatically, but not to a particular instance. Notice the following:

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

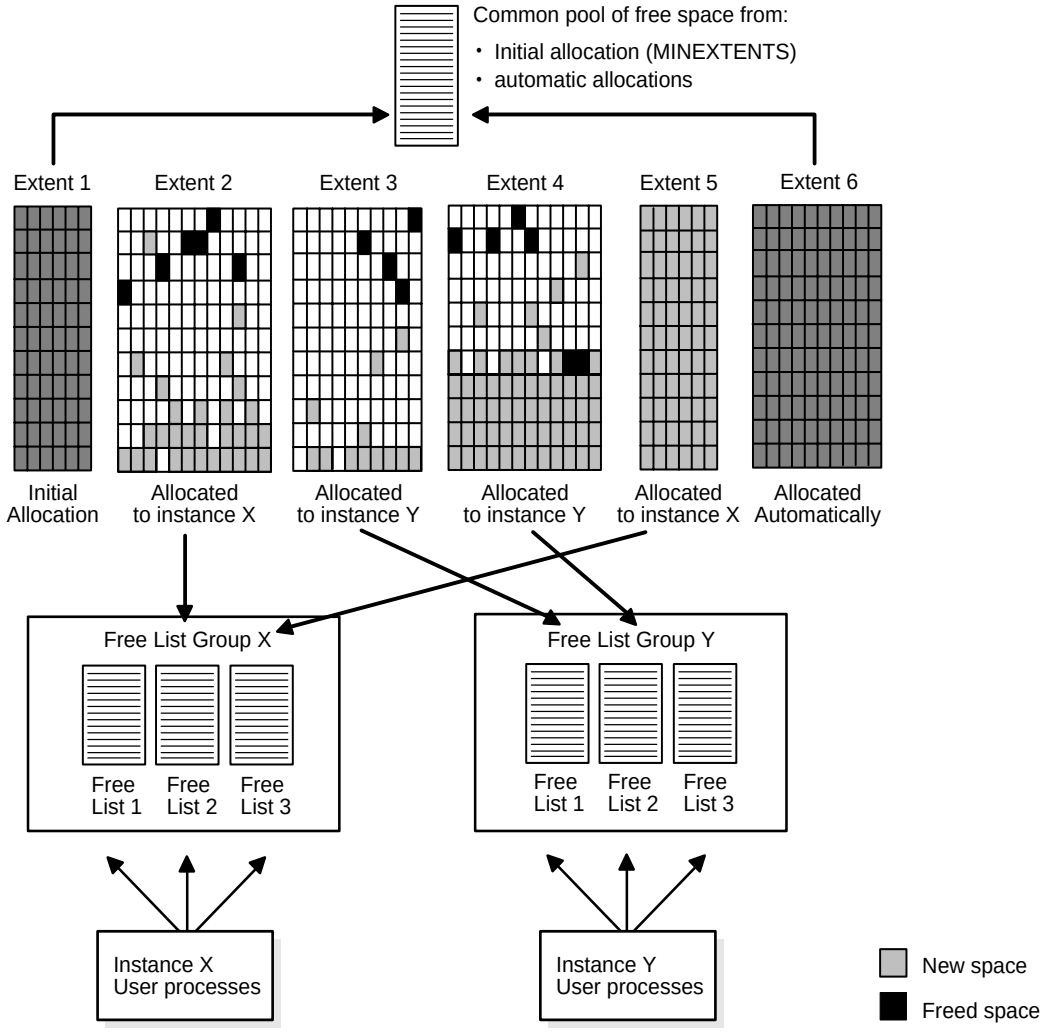
Each user process running on instance X uses one of the free lists in group X, and each user process on instance Y uses one of the free lists in group Y. If more instances start up, their user processes share free lists with instance X or Y.

Complex Free List Group Example

The simple case in Figure 7–5 becomes more complicated when you consider that extents are not permanently allocated to instances, and that space allocated to one instance cannot be used by another instance. Each free list group has its own master free list. After allocation, some blocks go onto the master free list for the group, some go to a process free list, and some do not belong to a free list at all. If the application is totally partitioned, then once blocks are allocated to a given instance, they stay with that instance. However, blocks can move from one instance to another if the application is not totally partitioned.

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

Figure 7–5 Groups of Free Lists for a Table



Partitioning Data with Free List Groups

In general, all tables should have the same number of free list groups, but the number of free lists within a group may vary, depending on the type and amount of activity of each table.

Partitioning free space can particularly improve the performance of applications that have a high volume of concurrent inserts, or updates requiring new space, from multiple instances. Performance improvements also depend, of course, on your operating system, hardware, data block size, and so on.

In a multi-instance environment, information about multiple free lists and free list groups is not preserved upon import. If you use Export and Import to back up and restore your data, it will be difficult to import the data so that it is partitioned again.

How Oracle Partitions Free List Groups

The actual free list group block is determined by hashing the Oracle process ID by the number of free list groups. For example, if there are 3 instances and 35 free list groups, then instance 1 handles the first twelve free list groups, instance 2 the next twelve, and instance 3 the remaining eleven.

Associating Instances, Users, and Locks with Free List Groups

This section describes:

- Associating Instances with Free Lists
- Associating User Processes with Free Lists
- Associating PCM Locks with Free Lists

Associating Instances with Free Lists

Data partitioning can reduce contention for data blocks. The PCM locks that often cover blocks in one free list group tend to be held primarily by the instance using that free list group. This is because an instance that modifies data is usually more likely to reuse that data than other instances. However, if multiple instances take free space from the same extent, they are more likely to contend for blocks in that extent if they subsequently modify the data that they inserted.

Assignment of New Instances to Existing Free List Groups

If MAXINSTANCES is greater than the number of free list groups in the table or cluster, then an instance number maps to the free list group associated with:

instance_number modulo *number_of_free_list_groups*

"Modulo" (or "rem" for "remainder") is a formula for determining which free list group should be used by calculating a remainder value. In the following example there are 2 free list groups and 10 instances. To determine which free list group instance 6 will use, the formula would read 6 modulo 2 = 0. Six divided by 2 is 3 with zero remainder, so instance 6 will use free list group 0. Similarly, instance 5 would use free list group 1 because 5 modulo 2 = 1. Five is divisible by 2 with a remainder of 1.

If there are more free list groups than MAXINSTANCES, then a different hashing mechanism is used. If multiple instances share one free list group, they share access to every extent specifically allocated to any instance sharing that free list group.

FREELIST GROUPS and MAXINSTANCES

In a system with relatively few nodes, the FREELIST GROUPS option for a table should generally have the same value as the MAXINSTANCES option of CREATE DATABASE, which limits the number of instances that can access a database concurrently. In a Massively Parallel Processing system, however, MAXINSTANCES could be many times larger than FREELIST GROUPS so that many instances share one group of free lists.

> **See Also:** *Oracle8i Parallel Server Administration, Deployment, and Performance* for more information on associating instances, users, and locks with freelist groups.

Associating User Processes with Free Lists

User processes associate with process free lists based on their Oracle process IDs. Each user process has access to only one free list in the free list group for the instance on which it is running. Every user process also has access to the master free list of free blocks.

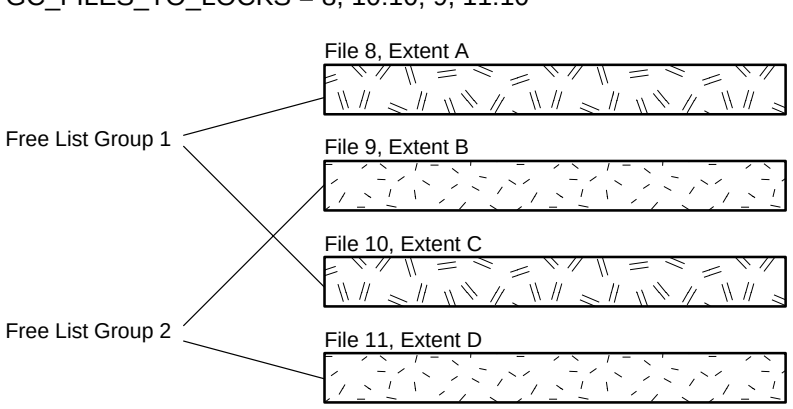
If a table has multiple free lists but does not have multiple free list groups, or has fewer free list groups than the number of instances, then each free list is shared by user processes from different instances.

Associating PCM Locks with Free Lists

If each extent in a table is in a separate data file, you can use the GC_FILES_TO_LOCKS parameter to allocate specific ranges of PCM locks to each extent, so that each set of PCM locks is associated with only one group of free lists.

Figure 7–6 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 PCM locks is associated with files 8 and 10, and a different set of PCM 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 7–6 Extents and Free List Groups

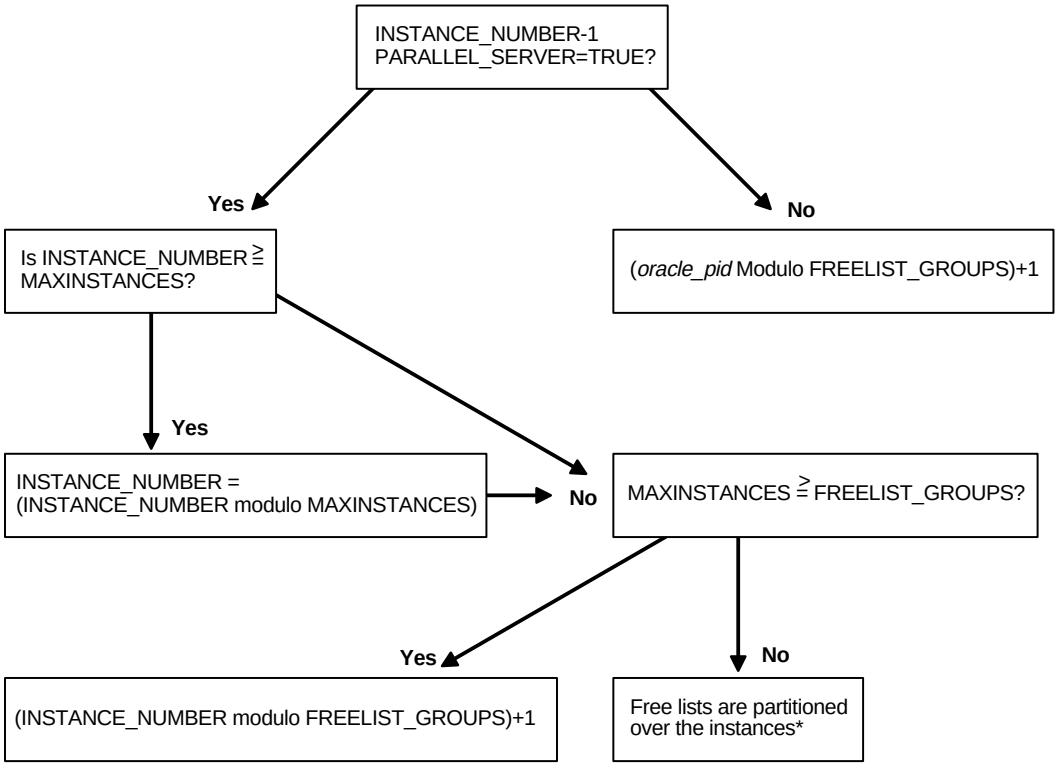


This example assumes total partitioning for reads as well as writes. If more than one instance is to update blocks, then it would still be desirable to have more than one lock per file to minimize forced reads and writes. This is because even with a shared lock, *all* blocks held by a lock are subject to forced reads when another instance tries to read even *one* of the locked blocks.

How Oracle Assigns Free Lists and Free List Groups to Instances

Figure 7–7 illustrates how free lists and free list groups are assigned to instances.

Figure 7–7 How Free Lists and Free List Groups Are Assigned



Using the statement ALTER SESSION INSTANCE_NUMBER, you can increase the instance number value beyond the value of MAXINSTANCES. Figure 7–7 shows how this is taken into account: for the purposes of the internal calculation whereby free list groups are assigned, the instance number is brought back within the boundaries of MAXINSTANCES.

SQL Options for Managing Free Space

Several SQL options enable you to allocate free lists and free list groups for tables, clusters, and indexes. You can explicitly specify that new space for an object be taken from a specific data file. You can also associate free space with particular free list groups that you can then associate with particular instances.

The SQL statements include:

```
CREATE [TABLE | CLUSTER | INDEX]
  STORAGE
  FREELISTS
  FREELIST GROUPS
ALTER [TABLE | CLUSTER | INDEX]
  ALLOCATE EXTENT
  SIZE
  DATAFILE
  INSTANCE
```

You can use these SQL options with the initialization parameter INSTANCE_NUMBER to associate data blocks with instances.

> **See Also:** *Oracle8i SQL Reference* for complete syntax of these statements.

Controlling Extent Allocation

This section covers the following topics:

- Automatic Allocation of New Extents
- Pre-allocation of New Extents
- Dynamic Allocation of Blocks on Lock Boundaries

When a row is inserted into a table and new extents need to be allocated, a certain number of contiguous blocks, as specified by !*blocks* in the GC_FILES_TO_LOCKS parameter, is allocated to the free list group associated with an instance. Extents allocated when the table or cluster is first created and new extents that are automatically allocated, add their blocks to the master free list or to the space above the high water mark.

Automatic Allocation of New Extents

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

Pre-allocation of New Extents

You have two options for controlling the allocation of new extents.

- Pre-allocating Extents to Free List Groups
- Dynamic Allocation of Blocks on Lock Boundaries

Pre-allocating Extents to Free List Groups

Pre-allocating extents is a static approach to the problem of preventing automatic allocation of extents by Oracle. You can pre-allocate extents to tables that have free list groups. This means that all free blocks are formatted into free lists, which will reside in the free list group of the instance to which you are pre-allocating the extent. This approach is useful if you need to partition data so as to greatly reduce all pinging on insert, or if you need to accommodate objects that you expect will grow in size.

Note: You cannot completely eliminate false pinging.

Dynamic Allocation of Blocks on Lock Boundaries

To accommodate growth, the strategy of dynamically allocating blocks to free list groups is more effective than pre-allocating of extents. You can 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 pinging on the segment header. Instead, this method allocates blocks as needed so you do not have to pre-allocate 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.

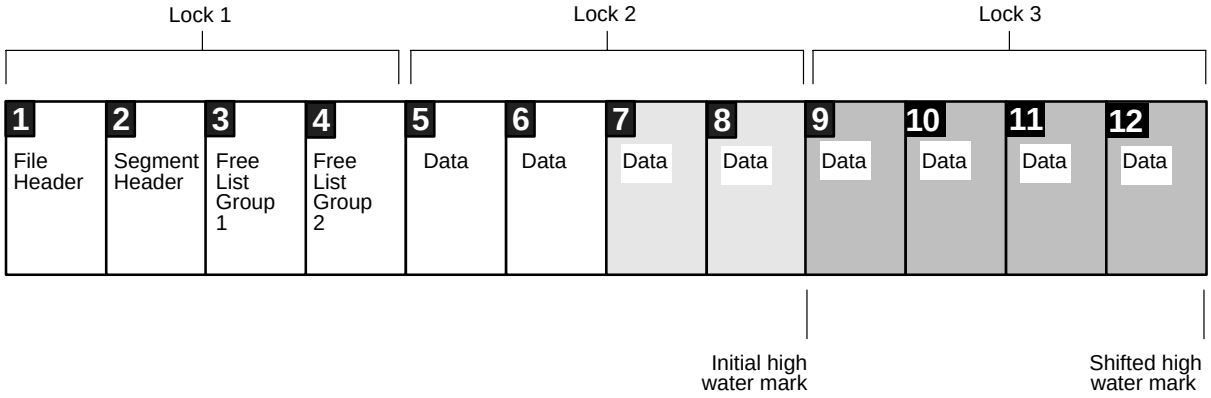
Using this method, you can either explicitly allocate the !*blocks* value, or leave the balance of new blocks still covered by the existing PCM lock. If you choose the latter, remember there still may be contention for the existing PCM lock by allocation to other instances. If the PCM lock covers multiple groups of blocks, there may still be unnecessary forced reads and writes of all the blocks covered by the lock.

Moving the High Water Mark of a Segment

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 following example with 4 blocks per lock (!4). Locks have been allocated before the block content has been entered. If we have filled data block D2, held by Lock 2, and then allocated another range of 4 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 7–8 A File with a High Water Mark That Moves as Blocks Are Allocated



Example This example assumes that GC_FILES_TO_LOCKS has the following setting for both instances:

```
GC_FILES_TO_LOCKS = "1000!5"
```

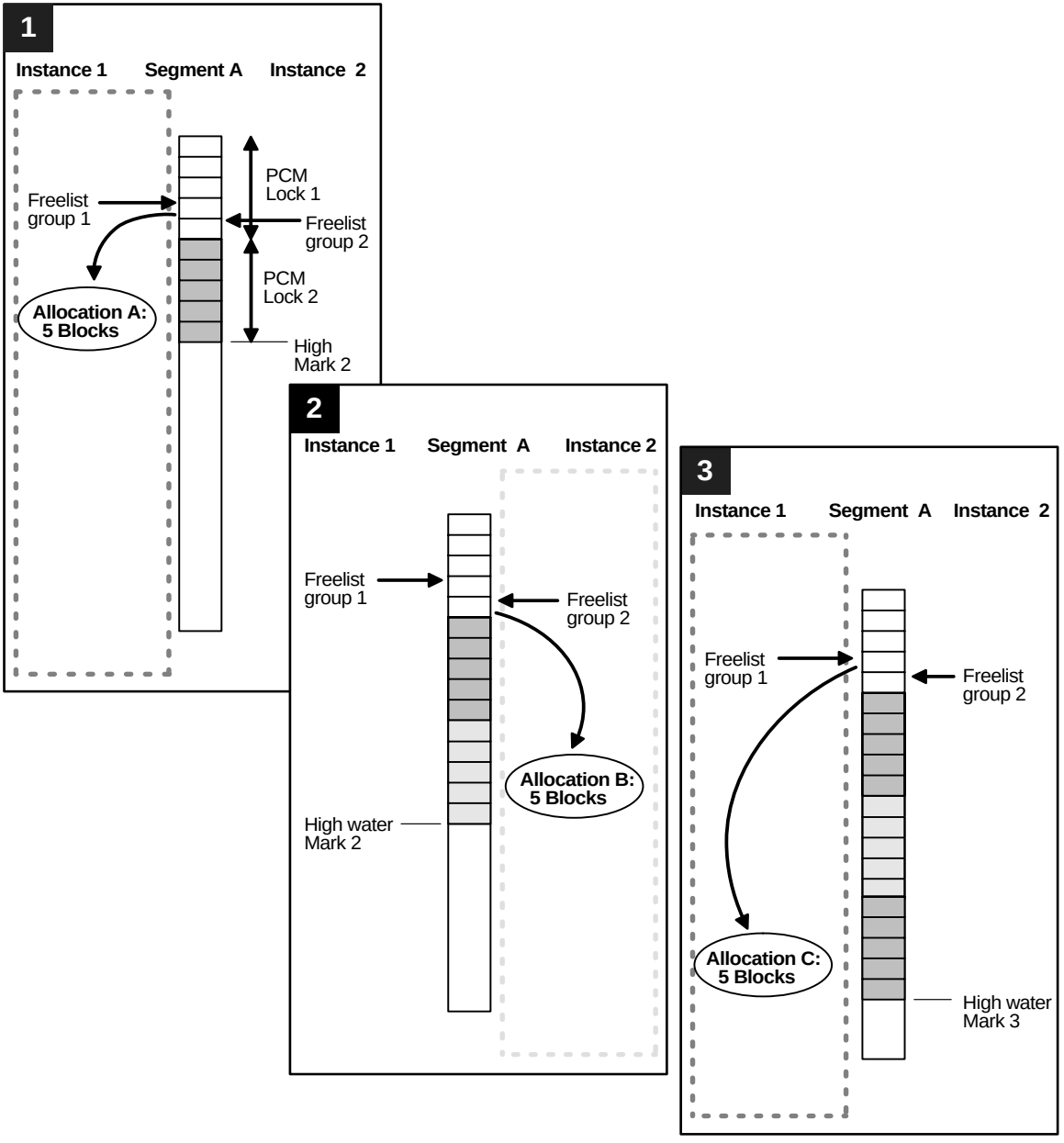
With the EACH option specified, each file in *file_list* is allocated *#locks* number of PCM locks. Within each file, !*blocks* specifies the number of contiguous data blocks to be covered by each lock.

Figure 7–9 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 7–9 Allocating Blocks within an Extent



8

Scalability and Oracle Parallel Server

This chapter describes the scalability features of Oracle Parallel Server. Topics in this chapter include:

Scalability Features of Oracle Parallel Server

You can implement Oracle Parallel Server using several features that enhance application performance and maintain high availability levels. These features:

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

This section discusses these features in four parts:

- Overview of Client-to-Server Connectivity
- Enhanced Scalability Using the Multi-Threaded Server
- The Four Levels of Scalability

The Oracle Database Configuration Assistant automatically configures most of these features for you. To manually configure these features, refer to the Net8 Administrator's Guide where noted in the following discussions.

Enhanced Throughput: Scale-Up

If tasks can run independently of one another, Oracle Parallel Server can distribute them to different nodes. This permits you to achieve scaleup: more processes run through the database in the same amount of time. The number of nodes used, however, depends upon the purpose of the system.

If processes can run faster, then the system accomplishes more work. The parallel execution feature, for example, permits scaleup: a system might maintain the same response time if the data queried increases tenfold, or if more users can be served. Oracle Parallel Server without the parallel execution feature also provides scaleup, but by running the same query sequentially on different nodes.

With a mixed workload of DSS, OLTP, and reporting applications, you can achieve scaleup by running multiple programs on different nodes. You can also achieve speed-up by rewriting the batch programs and separating them into a number of parallel streams to take advantage of the multiple CPUs.

Oracle Parallel Server also offers improved flexibility by overcoming memory limitations so you can use it to accommodate thousand of users. You can allocate or

deallocate instances as necessary. For example, as database demand increases, you can temporarily allocate more instances. Then you can deallocate the instances and use them for other purposes once they are no longer required. This feature is useful in internet-based computing environments.

Speed-Up and Scale-up: The Goals of Parallel Processing

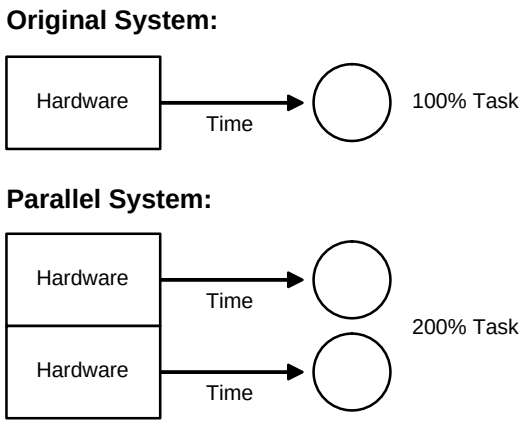
You can measure the performance goals of parallel processing in two ways:

- Scale-Up
- Speed-Up

Scale-Up

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

Figure 8–1 Scale-Up



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

You can measure scaleup using this formula:

$$Scaleup = \frac{Volume\_Parallel}{Volume\_Original}$$

Where:

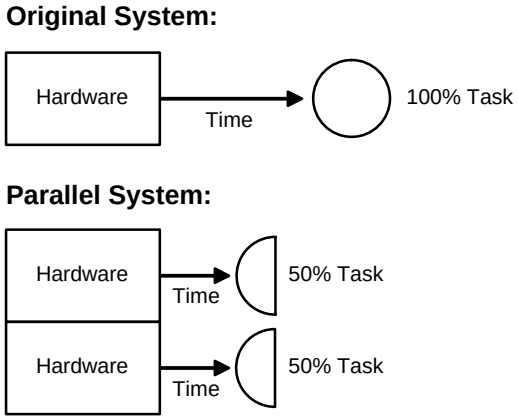
Volume_Original	is the transaction volume processed in a given amount of time on a small system.
Volume_Parallel	is the transaction volume processed in a given amount of time on a parallel system.

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

Speed-Up

Speed-up is the extent to which more hardware can perform the same task in less time than the original system. With added hardware, speed-up holds the task constant and measures time savings. Figure 8–2 shows how each parallel hardware system performs half of the original task in half the time required to perform it on a single system.

Figure 8–2 Speed-Up



However, you may not experience direct, linear speed-up. Instead, speed-up may be more logarithmic. That is, assume the system can perform a task of size "x" in a time duration of "n". But for a task of size 2x, the system may require a time duration of 3n.

> **Note:** For most OLTP applications, no speed-up can be expected; only scale-up. The overhead due to synchronization can in fact, cause slow-down.

When Is Parallel Processing Advantageous?

This section describes applications that commonly benefit from a parallel server.

- Decision Support Systems
- Applications that Update Different Data Blocks
- Departmentalized Applications

Decision Support Systems

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

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

Decision support applications are good candidates for Oracle Parallel Server because they only occasionally modify data, as in a database of financial transactions that is mostly accessed by queries during the day and is updated during off-peak hours.

Applications that Update Different Data Blocks

Applications that either update different data blocks or update the same data blocks at different times are also well suited to the parallel server. An example is a time-sharing environment where users each own and use one set of tables.

An instance that needs to update blocks held in its buffer cache must hold one or more instance locks in exclusive mode while modifying those buffers. Tune parallel server and the applications that run on it to reduce this type of contention for instance locks. Do this by planning how each instance and application uses data and partition your tables accordingly.

OLTP with Partitioned Data

Online transaction processing applications that modify different sets of data benefit the most from parallel server. One example is a branch banking system where each branch (node) accesses its own accounts and only occasionally accesses accounts from other branches.

OLTP with Random Access to a Large Database

Applications that access a database in a mostly random pattern also benefit from parallel server. This is true only if the database is significantly larger than any node's buffer cache. One example is a motor vehicle department's system where individual records are unlikely to be accessed by different nodes at the same time. Another example is an archived tax record or research data system. In cases like these, most access results in I/O even if the instance had exclusive access to the database. Oracle features such as 1:1 locking further improve performance of such applications.

Departmentalized Applications

Departmentalized applications are applications that you have effectively partitioned based on business or departmental processes. Such applications are also suitable for Oracle Parallel Server because they primarily modify different tables in

the same database. An example is a system where one node is dedicated to inventory processing, another is dedicated to personnel processing, and a third is dedicated to sales processing. In this case there is only one database to administer, not three.

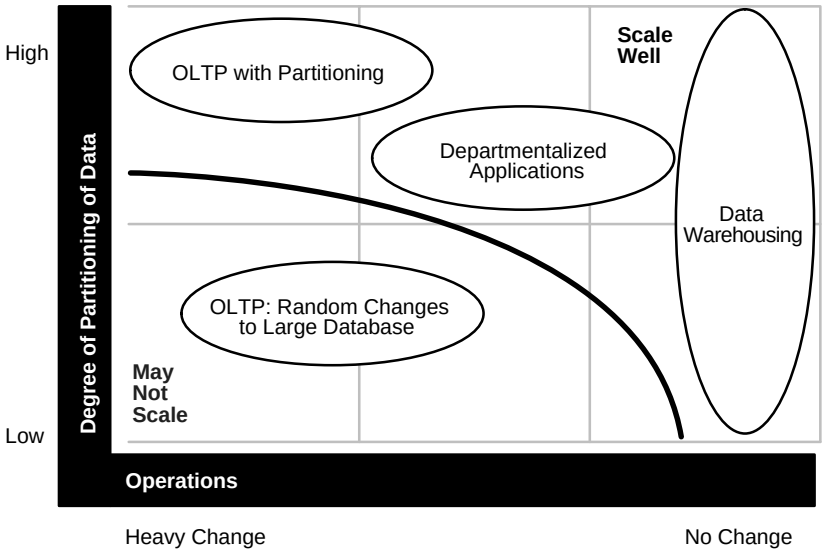
Application Profiles

Online transaction processing (OLTP) applications tend to perform best on symmetric multiprocessors; they perform well on clusters and MPP systems if they can be well partitioned. Decision support (DSS) applications tend to perform well on SMPs, clusters, and massively parallel systems. Select the implementation providing the power you need for the application(s) you require.

Figure 8–3 illustrates the relative scalability of different application types. Data warehousing applications, depicted by the left-most bubble, typically scale well since updates are less common and the degree of partitioning is higher than other application types. OLTP and departmentalized applications with partitioning and increasing rates of change also scale well.

OLTP applications that make random changes to large databases were historically not considered good parallel server candidates. Such applications, however, are becoming more scalable because they use advanced inter-node communication channels such as an interconnect. This is particularly true if, for example, a table is modified on one instance and then another instance reads the table. Such configurations are now much more scalable than in previous releases.

Figure 8–3 Scalability of Applications



Multi-Node Parallel Execution

You can execute operations in parallel using multi-node parallel execution. This can significantly reduce the duration of complex queries and DML statements.

Overview of Client-to-Server Connectivity

In Oracle Parallel Server, client-to-server connections are established using several subcomponents. A client submits a connection request to a listener process that resides on the destination node in the cluster. A "listener" process is a process that manages in-coming connection requests. The listener grants a client-to-server connection by way of a particular node based on several factors depending on how you configure the connection options.

Enhanced Scalability Using the Multi-Threaded Server

The Multi-Threaded Server allows enhanced scalability in that for a given amount of memory it allows you to support a larger number of users than when using dedicated servers. The incremental costs of memory as you add a user is less than

when using a dedicated server. In addition to improving the scalability of the number of users, Multi-Threaded Server offers additional benefits.

An increasing number of Oracle8*i* features require the Multi-Threaded Server. You need the Multi-Threaded Server if you use certain database features such as:

- Oracle8*i* JServer
- Connection Pooling
- Connection Concentration Feature

> **See Also:** *Net8 Administrator's Guide* for Multi-Threaded Server configuration information for Oracle Parallel Server environments.

Service Registration and the Multi-Threaded Server Functionality

An important feature of the Multi-Threaded Server is "Service Registration". This feature provides the listener with the service names, instance names and network addresses of the database, as well as the current state and load of all instances and MTS dispatchers. With this information, the listener can forward client connection requests to the appropriate dispatchers and dedicated servers.

Because this information is registered with the listener, you do not need to configure the `listener.ora` file with this static information about a database service. Service registration also extends benefits to connection load balancing which is a feature of the Multi-Threaded Server.

Connection Load Balancing

The connection load balancing feature provides exceptional benefits in Oracle Parallel Server environments where there are multiple instances and dispatchers. Connection load balancing improves performance by balancing the number of active connections among various instances and MTS 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 regardless of their location. This way, a listener can send an incoming client request for a specific service to the least loaded instance and least loaded dispatcher.

Service Registration also facilitates connect-time failover and client load balancing which you can use with or without MTS.

Connect-Time Failover for Multiple Listeners

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

Client Load Balancing for Multiple Listeners

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

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

> **See Also:** For configuration information on the features described in this section, please refer to the *Net8 Administrator's Guide.*

The Four Levels of Scalability

Successful implementation of parallel processing and parallel database requires optimal scalability on four levels:

- Scalability of Hardware and Network
- Scalability of Operating System
- Scalability of Database Management System
- Scalability of Application

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

Scalability of Hardware and Network

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

Bandwidth and Latency

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

Hardware scalability depends heavily on very low latency. Lock coordination traffic communication is characterized by a large number of very small messages among the LMD processes.

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

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

Disk Input and Output

Local I/Os are faster than remote I/Os (those which occur between nodes). If a great deal of remote I/O is needed, the system loses scalability. In this case you can partition data so that the data is local.

Note: Various clustering implementations are available from different hardware vendors. On shared disk clusters with dual ported controllers, the latency is the same from all nodes. However, with MPP (shared nothing) systems, this may not me true.

Scalability of Operating System

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

Software scalability can be an important issue if one node is a shared memory system (that is, a system where multiple CPUs connect to a symmetric

multiprocessor single memory). Methods of synchronization in the operating system can determine the scalability of the system. In asymmetrical multiprocessing, for example, only a single CPU can handle I/O interrupts. Consider a system where multiple user processes request resources from the operating system:

Scalability of Database Management System

An important distinction in parallel server architectures is internal versus external parallelism; this has a strong effect on scalability. The key difference is whether the object-relational database management system (ORDBMS) parallelizes the query, or an external process parallelizes the query.

Disk affinity can improve performance by ensuring that nodes mainly access local, rather than remote, devices. An efficient synchronization mechanism enables better speed-up and scaleup.

See Also:

- *Oracle8i Parallel Server Administration, Deployment, and Performance*
- *Oracle8i Designing and Tuning for Performance*

Scalability of Application

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

Note: Applications must be specifically designed to be scalable!

No matter how scalable the hardware, software, and database may be, a table with only one row which every node is updating will synchronize on one data block. Consider the process of generating a unique sequence number:

```
UPDATE ORDER_NUM
SET NEXT_ORDER_NUM = NEXT_ORDER_NUM + 1;
COMMIT;
```

Every node needing to update this sequence number must wait to access the same row of this table: the situation is inherently unscalable. A better approach is to use sequences to improve scalability:

```
INSERT INTO ORDERS VALUES
   (order_sequence.nextval, ... )
```

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

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

See Also: *Oracle8i Parallel Server Administration, Deployment, and Performance* for information on designing databases and application analysis.

The Sequence Generator

Oracle Parallel Server allows users on multiple instances to generate unique sequence numbers with minimal synchronization.

The sequence number generator allows multiple instances to access and increment a sequence without contention among instances for sequence numbers and without waiting for transactions to commit. Each instance can have its own sequence cache for faster access to sequence numbers. Distributed Lock Manager locks coordinate sequences across instances in Oracle Parallel Server.

This section describes the CREATE SEQUENCE statement and its options.

The CREATE SEQUENCE Statement

The SQL statement CREATE SEQUENCE establishes a database object from which multiple users can generate unique integers without waiting for other users to commit transactions to access the same sequence number generator.

Oracle Parallel Server allows users on multiple instances to generate unique sequence numbers with minimal cooperation or contention among instances. Instance locks coordinate sequences across instances in Oracle Parallel Server.

Sequence numbers are always unique, unless you use the CYCLE option. However, you can assign sequence numbers out of order if you use the CACHE option without the ORDER option, as described in the following section.

> **See Also:** *Oracle8i SQL Reference* for more information about the CREATE SEQUENCE and CYCLE options.

The CACHE Option

The CACHE option of CREATE SEQUENCE pre-allocates sequence numbers and retains them in an instance's System Global Area for faster access. You can specify the number of sequence numbers cached as an argument to the CACHE option. The default value is 20.

Caching sequence numbers significantly improves performance but can cause the loss of some numbers in the sequence. Losing sequence numbers is unimportant in some applications, such as when sequences are used to generate unique numbers for primary keys.

A cache for a given sequence is populated at the first request for a number from that sequence. After the last number in that cached set of numbers is assigned, the cache is repopulated with another set of numbers.

Each instance keeps its own cache of sequence numbers in memory. When an instance shuts down, cached sequence values that have not been used in committed DML statements can be lost. The potential number of lost values can be as great as the value of the CACHE option multiplied by the number of instances shutting down. Cached sequence numbers can be lost even when an instance shuts down normally.

The ORDER Option

The ORDER option of CREATE SEQUENCE guarantees that sequence numbers are generated in the order of the requests. You can use the ORDER option for time-stamp numbers and other sequences that must indicate the request order across multiple processes and instances.

If you do not need Oracle to issue sequence numbers in order, the NOORDER option of CREATE SEQUENCE can significantly reduce overhead in an Oracle Parallel Server environment.

Note: Oracle Parallel Server does not support the CACHE option with the ORDER option of CREATE SEQUENCE when the database is mounted in parallel mode. Oracle cannot guarantee an order if each instance has some sequence values cached. Therefore, if you should create sequences with both the CACHE and ORDER options, they will be ordered but not cached.

Oracle Parallel Execution on Oracle Parallel Server

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

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

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

Oracle parallel execution can run with or without the Oracle Parallel Server. Without the Oracle Parallel Server option, parallel execution cannot perform multi-node parallelism. Oracle Parallel Server optimizes the Oracle8*i* Enterprise Edition running on clustered hardware using a parallel cache architecture to avoid shared memory bottlenecks in OLTP and DSS applications.

9

High Availability and Oracle Parallel Server

This chapter describes the concepts and some of the "best practices" methodologies for using Oracle Parallel Server to implement high availability. This chapter includes the following topics:

What is High Availability?

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

When failures occur, a process known as "failover" moves processing performed by the failed component to the backup component. The failover process re-masters system-wide resources, recovers partial or failed transactions, and restores the system to normal, preferably within a matter of mircroseconds. The more transparent that failover is to the users, the higher the availability of the system.

Oracle Parallel Servers are inherently high availability systems. The clusters that are typical of Oracle Parallel Server environments, as described in Chapter 2, can provide continuous service for both planned and unplanned outages.

Measuring Availability

You can classify systems and evaluate their expected availability by system type. Mission- and business-critical applications such as mail and internet servers probably require a significantly greater availability than do less popular applications. As well, some systems may have a "24 x 7" uptime requirement, while others such as a stock market tracking system will have near 100% uptime requirements for specific timeframes, such as when the stock market is open.

The Metrics of High Availability

The software industry generally measures availability using two types of metrics:

- Mean time to recover (MTTR)
- Mean time between failures (MTBF)

For most failure scenarios, the industry focuses on MTTR issues and investigates how to optimally design systems to reduce these. MTBF is generally more applicable for hardware availability metrics; this chapter does not go into detail about MTBF. However, given that you can design Oracle Parallel Server clusters to avoid single points-of-failure, component failures may not necessarily result in application unavailability. Hence, Oracle Parallel Server can greatly reduce the MTBF from an application availability standpoint.

Another metric that is generally used is "number of nines". For example, 526 minutes of system unavailability per year results in 99.9% or "3-nines availability", and 5 minutes of system unavailability per year results in a 99.999% or "5-nines availability".

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

Causes of Outages

Downtime can be classified into two categories:

- Planned Downtime
- Unplanned Downtime

Planned Downtime

Scheduled maintenance, product upgrades, and application modifications are typically performed during these timeframes. End users generally do not have system access during these periods. Oracle Parallel Server provides many features to minimize the need for planned downtime to perform routine maintenance. These include failover to another node for system maintenance, online reorganization, online backups, and partitioned operations.

Unplanned Downtime

Unplanned downtime results when component and / or system failures result in a "down system". User error can also contribute to unplanned downtime. Such failures can be due to either hardware or software problems and can involve CPU, memory, operating system, the database, or the network.

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

Planning for High Availability

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

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

System Level Planning

System level planning involves:

- Capacity Planning
- Redundancy Planning

Capacity Planning

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

If an application runs on a single symmetric multi-processing (SMP) machine with single instance Oracle, a natural growth path is to migrate this database to a larger SMP machine. However, depending on your hardware vendor's product line, this may not be an option.

Oracle Parallel Server allows you to add nodes to your system to increase capacity and handle application growth. You can do this online without stopping the database, and with minimal interference to existing client transactions.

Redundancy Planning

Redundancy planning means duplicating system components such that no single component failure reduces system unavailability. Redundant components are often used in high-end SMP machines to protect against failures. For example, redundant power supplies and redundant cooling fans are not uncommon in high-end SMP server systems. A clustered Oracle Parallel Server environment can extend this redundant architecture to a higher level by creating complete redundancy such that there is no single point-of-failure.

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

Oracle Parallel Server and High Availability

Oracle Parallel Server builds higher levels of availability on top of the standard Oracle features. All single instance high availability features such as Fast-start Recovery and Online Re-organizations apply with Oracle Parallel Server as well. Fast-start Recovery can greatly reduce MTTR with minimal effects on online application performance. On-line Re-organizations reduce the durations of planned downtimes. Operations that used to be performed off-line during maintenance periods can now be performed on-line while users update the underlying objects. Oracle Parallel Server preserves all these standard Oracle features.

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

To configure Oracle Parallel Server for high availability, you must carefully consider the hardware and software component issues of your cluster as described under the following heading.

Cluster Components and High Availability

This section describes the high availability issues for the following cluster components:

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

See Also : Chapter 2 for more information about these components.

Cluster Nodes

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

Cluster Interconnects

Interconnect redundancy is often overlooked in clustered systems. This is because the Mean Time To Fail (MTTF) is generally several years and therefore cluster interconnect redundancy may not be a high priority. Also, depending on the system and sophistication level, a redundant cluster interconnect could be cost prohibitive and have insufficient business justification.

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

Storage Devices

Oracle Parallel Server operates on a single image of the data; all nodes in the cluster access the same set of data files. Database administrators are encouraged to use hardware based mirroring to maintain redundant media. In this regard, Oracle Parallel Server is no different from single instance Oracle. Disk redundancy depends on the underlying hardware and software mirroring in use, such as RAID.

Operating System Software and Cluster Managers

It was already mentioned that Oracle Parallel Server environments have full node redundancy. Each node runs its own operating system copy. Hence, the same considerations about node redundancy also apply to the operating system. The cluster manager is an extension of the operating system. Since the Cluster Manager software is also installed on all the nodes of the cluster, full redundancy is assured.

Database Software

In Oracle Parallel Server, the database binaries are installed on the local disks of each node and an instance runs on each node of the cluster. All instances have equal

access to all data and can process any transactions. In this way, Oracle Parallel Server ensures full database software redundancy.

Disaster Planning

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

Oracle offers other solutions such as standby database and replication to facilitate more comprehensive disaster recovery planning. You can use these solutions with Oracle Parallel Server where one cluster hosts the primary database and another remote system or cluster hosts the disaster recovery database. Oracle Parallel Server is not required on either site.

Failure Protection Validation

Once you have carefully considered the system level issues, validate that the Oracle Parallel Server environment protects against potential failures. The following is comprehensive but non-exhaustive list of causes of failures you can use for this process:

- Cluster Component
- CPU
- Memory
- Interconnect Software
- Operating System
- Cluster Manager
- Oracle Database Instance Media
- Corrupt/Lost Control File
- Corrupt/Lost Log File
- Corrupt/Lost Data File Human Error
- Dropped/Deleted a Database Object

As discussed under "System Level Planning", Oracle Parallel Server environments protect against cluster component failures and software failures. However, media failures and human error may still cause system "downtime". Oracle Parallel Server, as with single instance Oracle, operates on one set of files. For this reason, you should adopt best practices to avoid media failures.

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

Once you deploy your system, the key issue is the transparency of failover and its duration. The next section describes failover in more detail.

Failover and Oracle Parallel Server Systems

The following section describes the basics of failover and the various features Oracle Parallel Server offers to implement it in high availability systems. Topics in this section include:

- The Basics of Failover
- Client Failover
- Server Failover

The Basics of Failover

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

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

The Duration of Failover

The duration of failover includes the time a system requires to remaster system-wide resources and recover from failures. The duration of the failover process can be a relatively short interval on certified platforms. For existing users, failover entails both server and client failover actions. For new users, failover only entails the server failover time.

Client Failover

It is important is to hide system failures from database client connections. Such connections can include application users in client server environments or mid-tier database clients in multi-tiered application environments. When database failures occur, clients should not notice a loss of connection. Properly configured failover mechanisms transparently reroute client sessions to an available node in the cluster. This capability in the Oracle database is referred to as "Transparent Application Failover".

What Is Transparent Application Failover?

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

How Does Transparent Application Failover Work in Oracle Parallel Server?

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

Elements of Active Database Connections Affected by TAF

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

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

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

- Client-Server Database Connections
- Users' Database Sessions n Executing Commands
- Open Cursors Used for Fetching
- Active Transactions
- Server-Side Program Variables

Transparent Application Failover automatically restores some of these elements. Other elements, however, may need to be embedded in the application code to enable transparent application failover to recover the connection.

> **See Also:** *Net8 Administrator's Guide* for more information about configuring TAF.

Uses of Transparent Application Failover

While failing over client sessions during system failures is a strong benefit of Transparent Application Failover, there are other useful scenarios in which Transparent Application Failover improves system availability. These are:

- Transactional Shutdown
- Load Balancing

Transactional Shutdown

It is sometimes necessary to take nodes out of service for maintenance and / or repair. For example, you may want to apply patch releases without interrupting service to application clients. By using the TRANSACTIONAL clause of the SHUTDOWN statement, a node may be taken out of service such that the shutdown event is deferred until all existing transactions complete. In this way client sessions may be migrated to another node of the cluster at transaction boundaries.

Also, after performing a transactional shutdown, new transactions that are submitted get routed to an alternate node in the cluster. A SHUTDOWN IMMEDIATE is performed on the node when all existing transactions complete.

Load Balancing

A database is available when it processes transactions in a timely manner. When the load exceeds a node's capacity, the client transaction response times are adversely

affected and the database availability is compromised. It then becomes important to be able to manually migrate a group of client sessions to a less heavily loaded node to maintain response times for application availability.

Transparent Application Failover Restrictions

When a connection is lost, you will see the following effects:

- All PL/SQL package states on the server are lost at failover
- ALTER SESSION statements are lost
- If failover occurs when a transaction is in process, then each subsequent call causes an error message until the user issues an OCITransRollback call. Then Oracle issues an OCI success message. Be sure to check this message to see if you must perform additional operations.
- Continuing work on failed-over cursors may cause an error message
- If the first command after failover is not a SQL SELECT or OCIStmtFetch statement, an error message results
- Failover only takes effect if the application is programmed using OCI Release 8.0 or greater

Database Client Experience During Failover

The important issue during failover operations is the extent to which the failure is masked from existing client connections.

Query Clients At failover, in-progress queries are re-issued and processed from the beginning. This may extend the duration of the next query if the original query took a long time. With transparent application failover, the failure is masked for query clients with an increased response time being the only client observation. If the client query can be satisfied with data in the buffer cache of the surviving node that the client reconnected to, the increased response is minimal. Using the PRECONNECT method in transparent application failover further minimizes response time by saving the time to reconnect to a surviving Instance.

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

You can also use a callback function to notify clients of the failover so that the clients do not misinterpret the delay for a failure. This prevents the clients from manually attempting to re-establish their connections.

DML Clients For DML database clients that perform INSERT, UPDATE, and DELETE operations providing the application coded fully exploits the Oracle Call Interface (OCI) libraries, in-flight DML transactions on the failed instance may be restarted on a surviving instance without client knowledge. This achieves application failover, without manual reconnects, but requires application level coding. The coding necessary essentially handles certain Oracle error codes and performs a reconnect when those error codes are returned.

If the application coding is not in place, INSERT, UPDATE, and DELETE operations on the failed instance return an unhandled Oracle error code and the transaction must be resubmitted for execution. Upon re-submission, Oracle routes the client connections to a surviving instance. The client transaction then experiences a system pause until server-side recovery completes.

Server Failover

Server-side failover in Oracle Parallel Server is different from regular, host-based failover solutions that are available on many server platforms.

Host-Based Failover

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

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

1. Detecting failure by monitoring the heartbeat
2. Re-organizing cluster membership in the Cluster Manager
3. Transferring of disk ownership from the primary node to a secondary node

4. Re-starting application and database binaries
5. Performing application and database recovery
6. Re-establishing client connections to the failover node

Oracle Parallel Server Failover

Oracle Parallel Server provides very fast server-side failover. This is accomplished by Oracle Parallel Server's concurrent, active-active architecture, in other words, multiple Oracle instances are concurrently active on multiple nodes and synchronize access to the same database. All nodes have concurrent ownership and access to all disks. When one node fails, all other nodes in the cluster maintain access to all the disks; there is no disk ownership to transfer, and database application binaries are already loaded into memory.

Depending on the size of the database, the duration of failover can vary. The larger the database, or the greater the size of its data files, the greater the failover delta benefit to using Oracle Parallel Server. This is because transfer of disk ownership from the primary to the secondary instance in host-based failover environments is proportional to the number and size of files that need to be failed over. The additional cost of restarting the application / database binaries, in host-based failover environments, is a fixed cost and is proportional to the size of the application and database binaries and to the extent of the application initialization actions.

The previously discussed client failover section analyzes client failover behavior in the midst of failure scenarios. New client connections get routed to available nodes in the cluster and certain existing client connections on the failed node can be configured to transparently fail over. However, in order for database clients to begin processing transactions on the available nodes, Oracle Parallel Server needs to complete its server-side recovery actions.

How Does Oracle Parallel Server Failover Work?

The recovery actions necessary during a failed node in Oracle Parallel Server include the following:

- Detecting failure
- Re-organizing cluster membership
- Performing database recovery

Detecting Failure

Oracle Parallel Sever depends on the Cluster Manager software for failure detection because the Cluster Manager maintains the heartbeat functions. The time it takes for the Cluster Manager to detect that a node is no longer in operation is a function of a configurable heartbeat timeout parameter. You can configure this value on most systems; the default and is typically one minute in duration. This value is inversely related to the number of false alarms or false failure detections as the cluster might incorrectly determine that a node is failing due to transient failures if the timeout is set too low. When failure is detected, cluster re-organization occurs.

Re-organizing Cluster Membership

When a node fails, Oracle must alter its cluster membership status. This is known as a "cluster re-organization" and it usually happens quickly; its duration is proportional to the number of surviving nodes in the cluster. Oracle Parallel Server is dependent on the Cluster Manager software for this information.

Oracle Parallel Server's Distributed Lock Manager provides the Cluster Manager interfaces to the software and exposes the cluster membership map to the Oracle instances when nodes are added / deleted from the cluster. The Distributed Lock Manager's LMON process on each cluster node communicates with the Cluster Manager on the respective nodes and exposes that information to the respective Oracle instances.

LMON also provides another useful function: when a node is no longer a member of the cluster, the surviving nodes do not see evidence of that node in the cluster, such as messages or writes to shared disk. These LMON-provided services are also referred to as Cluster Group Services (CGS). When a failure causes a change in a node's membership status within the cluster, LMON initiates the recovery actions that include re-mastering of PCM locks and Instance recovery.

At this stage, the Oracle Parallel Server environment is in a state of system pause, and most client transactions suspend until the necessary recovery actions are complete.

Performing Database Recovery

The database recovery steps necessary in Oracle Parallel Server include:

- Re-mastering the PCM lock resources of the failed instance
- Instance recovery includes cache recovery and transaction recovery

When an instance fails, PCM lock resources on the failed instance need to be re-mastered on the surviving cluster nodes. This is also referred to as Distributed Lock Manager database rebuild.

Re-mastering PCM Lock Resources of The Failed Instance

The time required for re-mastering of locks is a function of the number of PCM locks in the lock database. This number in turn depends upon the size of the buffer caches. Each distinct database block in a buffer cache requires one PCM lock if you use 1:1 releasable locking. However, in the case of releasable locks, the lock resources may have already been released and may not need to be re-mastered. When you use 1:N fixed locks, the number of locks is determined by the initialization parameters as each lock covers multiple blocks.

During this phase, all lock information is discarded and each surviving instance re-acquires all the locks it held at the time of the failure The lock space is now distributed uniformly across the remaining n instances. For any lock request, there is a 1/n chance that the request will be satisfied locally and a (N-1)/n chance that the lock request will involve remote operations. In the case of one surviving instance all lock operations will be satisfied locally.

Once re-mastering of the failed Instance PCM locks are complete, the in-flight transactions of the failed Instance needs to be cleaned up. This is known as instance recovery.

Instance Recovery

Instance recovery requires that an active Oracle Parallel Server instance detects failure and performs the necessary recovery actions on behalf of the failed Oracle Parallel Server Instance. The first Oracle Parallel Server instance that detects the failure, by way of its LMON process, assumes control of recovering the failed instance by taking over the failed instance's redo log files and performs instance recovery actions. This is why the redo log files need to be on a shared device such as a shared raw logical volume or cluster file system.

Instance recovery is said to be complete when cache recovery - in other words, on-line redo log files of the failed Instance have been replayed - and transaction recovery -in other words, all uncommitted transactions of the failed Instance are rolled back - are completed. Since transaction recovery may be performed in a deferred fashion, client transactions can start processing when cache recovery is complete.

Cache Recovery

Cache recovery requires Oracle to replay the online redo logs of the failed instance. Oracle performs cache recovery in parallel, in other words, parallel threads of work are set in motion to replay the redo logs of the failed Oracle instance. It may be important that you keep the length of the time interval for redo log replay to a predictable duration. The Fast-start Recovery feature in Oracle8 provides this capability.

Fast-start recovery uses a parameter called FAST_START_IO_TARGET to provide fine-grained control over the amount of redo log replay necessary for instance recovery. Redo log replay is performed by continuous checkpointing mechanisms that maintain redo log tails of consistent sizes. Database clients are not necessarily aware of the server side recovery actions since the only client experience is a brief system pause. Setting FAST_START_IO_TARGET to a particular value is important in keeping the system pause interval within acceptable bounds during system failures.

Oracle provides non-blocking rollback capabilities, so full database access can start as soon as online log files are replayed. After cache recovery is complete, Oracle begins transaction recovery.

Transaction Recovery

Transaction recovery comprises rolling back all uncommitted transactions of the failed Instance. These are "in-progress" transactions that did not commit and that Oracle needs to roll back.

Oracle8 Fast-start Rollback performs rollback in the background as a deferred process. Oracle uses a multi-version read consistency technology to provide on-demand rollback of only the row blocked by dead transactions, so new transactions can progress with minimal delay. Since new transactions do not have to wait for the entire dead transaction to be rolled back, long-running transactions no longer affect database recovery time.

Oracle8 Fast-start Rollback rolls back dead transactions in parallel using a recovery coordinator to spawn many recovery processes. Single instance Oracle rolls back dead transactions using the CPU of one node.

Oracle Parallel Server provides cluster-aware Fast-start Rollback capabilities that use all the CPU nodes of the cluster to perform parallel rollback operations. Each cluster node spawns a recovery coordinator and recovery processes to assist with parallel rollback operations. The Fast-start Rollback feature is thus "cluster aware" because the database is aware of and utilizes all cluster resources for parallel rollback operations.

While the default behavior is to defer transaction recovery, you may choose to configure your system so transaction recovery completes before allowing client transactions to progress. In this scenario, Oracle Parallel Server's ability to parallelize transaction recovery across multiple nodes is a more visible user benefit.

Oracle Parallel Server High Availability Configurations

The following section discusses the following three high availability configurations provided by Oracle Parallel Server:

- Default N-node Parallel Server Configuration
- Basic High Availability Configuration
- Shared High Availability Node Configuration

Default N-node Parallel Server Configuration

The default N-node Oracle Parallel Server configuration is the default Oracle Parallel Server environment. Client transactions are processed on all nodes of the cluster and client sessions may be load balanced at connect time. Response time is optimized for available cluster resources, such as CPU and memory, by distributing the load across cluster nodes to create a highly available environment.

Benefits of N-Node Oracle Parallel Server Configurations

In the event of a node failure, an Oracle Parallel Server Instance on another node will perform the necessary recovery actions as previously discussed. The database clients on the failed Instance may be load balanced across the (N-1) surviving Instances of the cluster. The increased load on each of the surviving Instances may be kept to a minimum and availability increased by keeping maintaining response times within acceptable bounds. In this configuration, the database application workload may be distributed across all nodes and therefore provides high utilization of cluster machine resources.

Basic High Availability Configuration

You can easily configure Oracle Parallel Server into a basic high availability configuration; the primary instance on one node accepts user connections while the secondary instance on the other node only accepts connections when the primary node fails. While you can configure this manually by controlling the routing of transactions to specific instances, Oracle Parallel Server provides the Primary / Secondary Instance feature to accomplish this.

You configure the Primary / Secondary Instance feature by setting the `init`*`sid`*`.ora` parameter ACTIVE_INSTANCE_COUNT to 1. The instance that first mounts the database assumes the role of primary instance. The other instance assumes the role of secondary instance. If the primary instance fails, the secondary instance assumes the primary role. When the failed instance returns to active status, it assumes the secondary instance role.

The secondary instance becomes the primary instance only after the Cluster Manager informs it about the failure of the primary instance but before Distributed Lock Manager reconfiguration and cache and transaction recovery begin. The redirection to the surviving instance happens transparently; application programming is not required. Only minor configuration changes to the client connect strings are required.

In the Primary / Secondary Instance configuration, both instances run concurrently, like in any N-node Oracle Parallel Server environment. However, database application users only connect to the designated primary instance. The primary node masters all Distributed Lock Manager locks. This minimizes communication between the nodes and provides performance levels that are almost comparable to a regular, single node database.

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

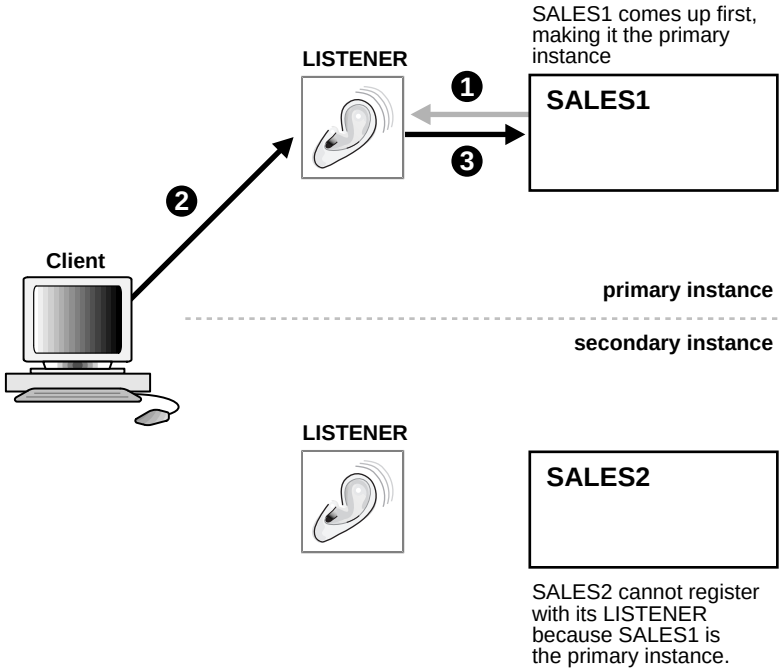
> **See Also :** *Oracle8i Parallel Server Setup and Configuration Guide* for information on configuring client connect strings.

The Primary / Secondary Instance feature works in both dedicated server and Multi-Threaded Server environments. However, it functions differently in each as described under the following headings.

Primary/Secondary Instance in Dedicated Server Environments

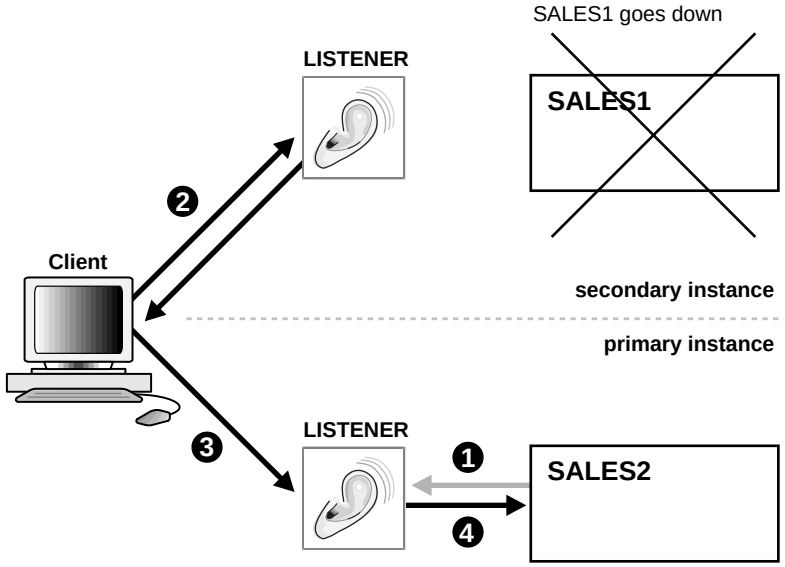
As shown in Figure 9–1, dedicated server environments do not have cross-instance listener registration. Therefore, a connection request made to a specific instance's listener can only be connected to that instance's service. This behavior is similar to default N-node Oracle Parallel Server clusters in dedicated server environments.

Figure 9–1 Primary/Secondary Instance Feature in Dedicated Server Environments



When the primary instance fails, the re-connection request from the client is rejected by the failed instance's listener. The secondary instance performs recovery and becomes the primary instance. Upon resubmitting the client request, the client re-establishes the connection using the new primary instance's listener that then connects the client to the new primary instance.

Figure 9–2 Node Failure in Dedicated Server Environments

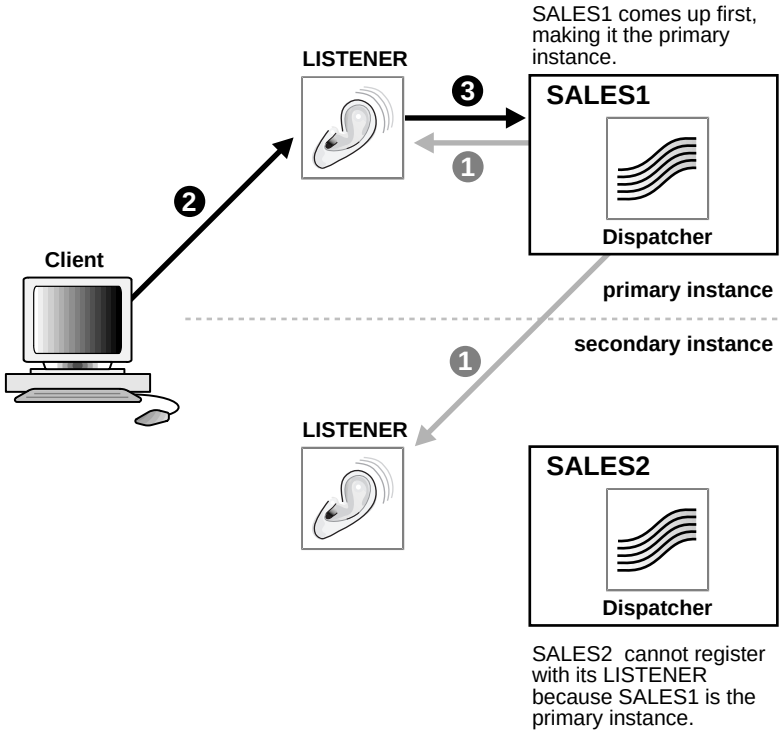


Primary/Secondary Instance and the Multi-Threaded Server

Oracle Parallel Server provides re-connection performance benefits when running in Multi-Threaded Server mode. This is accomplished by the cross-registration of all the dispatchers and listeners in the cluster.

In Primary / Secondary configurations, only the primary instance's dispatcher is aware of all listeners within the cluster, as shown in Figure , Step 1. A client may connect to either listener, Step 2–only the connection to the primary node listener is illustrated. The relevant listener then connects the client to the dispatcher as shown in Step 3–only the listener / dispatcher connection on the primary node is illustrated.

Figure 9–3 Primary/Secondary Instance Feature in Multi-Threaded Server Environments

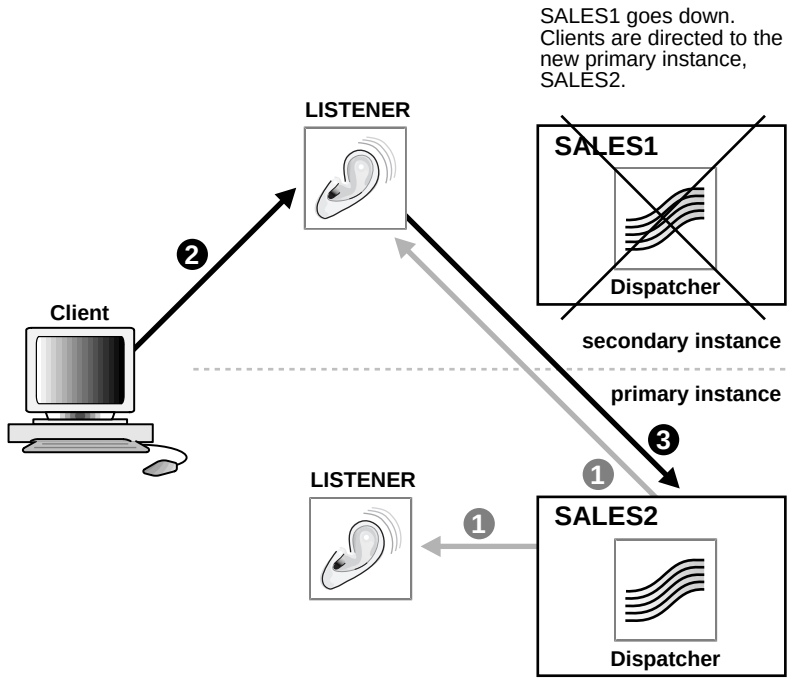


Specially configured clients can use the secondary instance for batch operations. For example, batch reporting tasks, index create operations can be performed on the secondary instance.

> **See Also :** *Oracle8i Parallel Server Setup and Configuration Guide* for instructions on how to connect to secondary instances.

In Figure 9–4, if the primary node fails, the dispatcher in the secondary instance register with the listener as shown in Step 1. When the client requests a reconnection to the database through either of the listeners, the listener directs the request to the secondary instance's dispatcher.

Figure 9–4 Node Failure in Multi-Threaded Server Environments



Benefits of Basic High Availability Configurations

There are a couple of different reasons for using this scenario instead of a default 2-node configuration. The Primary / Secondary Instance feature provides:

- A transition path for getting to an N-node configuration
- A highly available solution for applications that do not scale beyond one node

Transition Path to N-node Configurations

Using the Primary / Secondary configuration is a gradual way to migrate your application environment to an Oracle Parallel Server environment. Since all the client transactions are being performed on only one node at any given time, the Oracle Parallel Server tuning issues are minimized. Troubleshooting system problems is also simplified because you tune only one node at a time as opposed to simultaneously tuning two or more nodes.

Since availability is also dependent on the Database Administrator's ability to manage, tune, and troubleshoot the environment, the Oracle Parallel Server Primary / Secondary configuration provides a gradual way to ease the Database Administration staff into using Oracle Parallel Server.

Availability Solution for Applications That Do Not Scale

Applications may not scale beyond a single instance for a several reasons. The most common reason is application level serialization points. When the application design causes it to bottleneck on a single application resource, the application cannot scale beyond the capacity of that resource.

There may be other reasons why an application cannot scale. Update intensive and non-partitioned applications, for example, may not scale very well because of Oracle Parallel Server's disk-based synchronization mechanisms. In such cases, the cost of synchronization or block pinging can be excessive. However, the extensions to the Oracle8*i* Cache Fusion technology will make this issue one of diminishing importance to consider.

User environments that fit into one of the above two categories may favor the Oracle Parallel Server Primary / Secondary Configuration.

Shared High Availability Node Configuration

Running Oracle Parallel Server in an N-node configuration most optimally utilizes the cluster resources. However, as discussed previously, this is not always possible or advisable. On the other hand, the financial investment required to have an idle node for failover is often prohibitive. These situations may be instead best suited for a shared high availability node configuration.

This type of configuration would typically have several nodes with each running a separate application module or service where all application services share one Oracle Parallel Server database. You can set up a separate, designated node, as a failover node. While an Oracle Parallel Server instance is running on that node, no users are being directed to it during normal operation. In the event that any one of the application nodes fails, the workload can be directed to the high availability node.

While this configuration is a useful one to consider for applications that need to run on separate nodes, it works best if a middle tier application or Transaction Processing Monitor can direct the appropriate application users to the appropriate nodes. Unlike, the Primary / Secondary configuration, there is no database setup that automates the workload transition to the high availability node. The application, or mid-tier software, would need to be responsible for directing the users of the failed application node to the designated high availability node. The application would also need to control failing back the users once the failed node is operational. Failing back would free the failover node for processing user work from subsequent node failures.

Benefits of Shared High Availability Node Configurations

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

Toward Deploying High Availability

Oracle Parallel Server on clustered systems provides a fully redundant environment that is extremely fault resilient. Central to this high availability model is the Oracle Parallel Server architecture; all cluster nodes have an active instance that has equal access to all the data. If any node fails, all users have access to all data by way of surviving instances on the other cluster nodes. In-flight transactions on the failed node will be recovered by the first node that detects the failure. In this way, there is minimal interruption to end-user application availability with Oracle Parallel Server.

Part IV

Reference

Part IV includes the following reference information:

- Appendix A, "Differences Between Releases"
- Appendix B, "Restrictions"

A

Differences Between Releases

This appendix describes differences in Oracle Parallel Server from release to release.

> **See Also:** *Oracle8i Migration* for instructions on upgrading your database.

Differences Between 8.1 and 8.1.6

New Features

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

Obsolete Parameters

The LM_PROCS parameter is obsolete.

Obsolete Statistics

The following statistics are obsolete:

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

New Statistics

The following statistics are new:

- "global cache cr block send time" - total time to send the block.
- "global cache cr block log flushes" - number of log flushes.
- "global cache cr block log flush time" - total time log flushes took.
- "global cache prepare failures" - number of prepares which failed.

Changes in Default Parameter Settings

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

LM_LOCKS and LM_RESS Automatically Set by Oracle

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

Differences Between 8.0.4 and 8.1

Cache Fusion Architecture Changes

When one instance requests a consistent-read (CR) on a block held by another instance, Cache Fusion processing sends a CR copy of the requested block directly to the requesting instance by way of the interconnect. This greatly reduces cache coherency contention among instances during read / write conflicts.

Implementation of Cache Fusion requires that some background and foreground processes, namely LMON and LCK, now communicate directly from one instance to another over the interconnect. A new process, the Block Server Process (BSP), rolls back uncommitted transactions and copies CR server blocks for transmission to requesting instances. This reduces the pinging required to maintain cache coherency, thereby greatly improving performance.

Cache Fusion makes deployment of Oracle Parallel Server on OLTP and hybrid applications more feasible. Historically, databases that experienced random changes were not considered good parallel server candidates. With the advent of Cache Fusion and advanced cross-instance interconnect technology, OLTP and hybrid applications are becoming more scalable. This is particularly true if, for example, a table is modified on one instance and then another instance reads the table.

New Views

The following views are new:

- V$DLM_ALL_LOCKS view is new and shows statistics on locks whether they are blocking or blocked locks as well as all other lock types.
- V$DLM_RESS view is new and shows all resources associated with a lock according to lock type.
- V$DLM_CONVERT_LOCAL view is new and shows lock conversion statistics for locks opened on the local node.
- V$DLM_CONVERT_REMOTE view is new and shows lock conversion statistics for locks opened on remote nodes.
- V$DLM_MISC view is new and shows DLM message information.

Removal of GMS

For 8.1, the functionality of the GMS (Group Membership Services) has been moved from the GMS module to the vendor-specific Cluster Managers (CM) and the Oracle database kernel. In 8.1, a discrete GMS module is no longer visible to the Oracle user.

This change greatly improves vendor hardware compatibility with Oracle. From the user point-of-view, it also simplifies CM use and maintenance. The CM now starts automatically upon instance startup; you no longer need to manually startup and shut down member services.

Parallel Transaction Recovery is now "Fast-Start Parallel Rollback"

The name of the feature "Parallel Transaction Recovery" is now called "Fast-Start Parallel Rollback." In addition to the name change, in 8.0, SMON serially processed rollback segment recovery. This lead to extended rollback recovery periods. In 8.1, Fast-start parallel rollback reduces recovery time thus making the database available sooner. Parallel rollback uses multiple processes to recover rollback segments when the value for the parameter FAST_START_PARALLEL_ ROLLBACK, previously known as PARALLEL_TRANSACTION_RECOVERY, is greater than one.

The default for this parameter is LOW, implying that parallel recovery will use no more than 2 time the CPU_COUNT number of processes, in addition to SMON, to do parallel recovery.

To determine a more accurate setting, examine the contents of two new tables, V$FAST_START_SERVERS and V$FAST_START_TRANSACTIONS. Also consider the average time required to recover a transaction and your desired recovery period duration. When you set FAST_START_PARALLEL_ROLLBACK to a value greater than one, SMON starts multiple recovery processes to accommodate the amount of unrecovered rollback segments in the rollback file. The quantity of processes SMON starts is limited by the value for FAST_START_PARALLEL_ROLLBACK.

Changes to Instance Registration

The single name previously used to identify a service (SID) is replaced by three levels of addressing. The new parameters for instance registration are:

SERVICE_NAME	Name of highest level view of the service, specified in TNSNAMES.ORA. May span instances or nodes.

SERVICE_NAMES	Instance name of the service that can span several nodes. This parameter is specified in INIT.ORA
INSTANCE_NAME	Name of mid-level tier of the service. Corresponds to the ORACLE_SID of an instance.

Clients can connect to the service without specification of which handler or instance they require, thus allowing automatic load balancing to select the optimal handler in the optimal instance. Load balancing is discussed under the following heading.

Listener Load Balancing

The TNS listener now performs load balancing over distributed services spanning multiple nodes. The service, instance, and handler names are used to determine the load balancing behavior.

1. A client program specifies the name of the service it wants to connect to.
2. The listener finds the least loaded instance in the service.
3. The listener finds the least loaded handler in the instance.
4. The listener redirects the client to the optimal handler.

Diagnostic Enhancements

Oradebug is a utility used by consulting and support personnel to diagnose and troubleshoot problematic systems at runtime. Oradebug functionality is extended for the Oracle Parallel Server.

Oracle Parallel Server Management (OPSM)

OPSM is an option that simplifies parallel server administration. OPSM's 8.1 enhancements provide a single generic interface for administering parallel servers on any platform.

For more information about OPSM, see the *Oracle Parallel Server Management Users Guide.*

Parallel Server Installation and Database Configuration

The Oracle Universal Installer and Oracle Database Configuration Assistant are both cluster aware. In release 8.1, only a single installer session is required to install Oracle Parallel Server. The installer collects node information from the user,

distributes the required Oracle products to the specified nodes, and invokes the Oracle Parallel Server Assistant to set up the instances and create the database.

When Oracle Parallel Server Assistant is done with this process, the parallel server is available on all nodes and the parallel server configuration information is saved so that OPSM can use it to manage the new parallel server.

Instance Affinity for Jobs

Instance affinity for jobs is the association of jobs to an instance. Using the new DBMS_JOB package, you can indicate whether a particular instance, or any instance, can execute a user submitted job in the Oracle Parallel Server environment.

Use this release 8.1 feature to improve load balancing and limit block pinging. For instance, using Oracle Parallel Server and replication at the same time may result in block pinging problems on the deferred transactions queue if all instances in a clustered environment decide to propagate transactions from the deferred transaction queue. By limiting activity against tables to only one instance within a parallel server cluster, you can limit pinging. For more information, also see the *Oracle8i Supplied PL/SQL Packages Reference.*

Obsolete Parameters

The following parameters are obsolete as of release 8.1:

- GC_LCK_PROCS
- GC_LATCHES
- PARALLEL_DEFAULT_MAX_INSTANCES
- LOG_FILES
- OPS_ADMIN_GROUP
- CACHE_SIZE_THRESHOLD
- OGMS_HOME
- ALLOW_PARTIAL_SN_RESULTS
- SEQUENCE_CACHE_ENTRIES

Differences Between Release 8.0.3 and Release 8.0.4

New Initialization Parameters

The following initialization parameters were added for Oracle Parallel Server:

- OGMS_HOME
- GC_LATCHES
- PARALLEL_SERVER

Obsolete Initialization Parameters

The following initialization parameters are obsolete:

- MTS_LISTENER_ADDRESS
- MTS_MULTIPLE_LISTENERS

Obsolete Startup Parameters

- PARALLEL
- EXCLUSIVE

Dynamic Performance Views

The following view has changed:

- V$DLM_LOCKS

Group Membership Services

A new option has been added for the OGMSCTL command.

Differences Between Release 7.3 and Release 8.0.3

New Initialization Parameters

The following parameters were added specifically for Oracle Parallel Server:

- FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY
- LM_LOCKS
- LM_PROCS
- LM_RESS
- INSTANCE_GROUPS
- PARALLEL_INSTANCE_GROUP
- OPS_ADMIN_GROUP
- ALLOW_PARTIAL_SN_RESULTS

Obsolete GC_* Parameters

The following global cache lock initialization parameters are obsolete:

- GC_DB_LOCKS parameter
- GC_FREELIST_GROUPS parameter
- GC_ROLLBACK_SEGMENTS parameter
- GC_SAVE_ROLLBACK_LOCKS parameter
- GC_SEGMENTS parameter
- GC_TABLESPACES parameter

Changed GC_* Parameters

The values set by the GC_* parameters are not adjusted to prime numbers, but rather are left exactly as entered.

The following parameters have changed:

- GC_FILES_TO_LOCKS
- GC_ROLLBACK_LOCKS
- GC_RELEASABLE_LOCKS

Dynamic Performance Views

The following views are new:

- V$RESOURCE_LIMIT
- V$DLM_CONVERT_LOCAL
- V$DLM_CONVERT_REMOTE
- V$DLM_LATCH
- V$DLM_MISC
- V$FILE_PING
- V$CLASS_PING

The following views changed:

- V$BH
- V$SESSIONS
- V$SYSSTAT

Global Dynamic Performance Views

Global dynamic performance views (GV$ fixed views) were added, corresponding to each of the V$ views except for V$ROLLNAME.

Distributed Lock Manager

Oracle Parallel Server release 8.0 is not dependent on an external Distributed Lock Manager. The lock management facility is now internal to Oracle. The Integrated Distributed Lock Manager is dependent on an external node monitor.

LMON and LMD*n* processes have been added.

Instance Groups

The ability to logically group instances together and perform operations upon all of the associated instances was added.

Group Membership Services

Group Membership Services (GMS) is used by the Lock Manager (LM) and other Oracle components for inter-instance initialization and coordination.

Fine Grain Locking

In Oracle Parallel Server release 8.0, fine grain locking is available on all platforms. It is enabled by default.

Client-Side Application Failover

Oracle8 supports the ability of the application to automatically reconnect if the connection to the database is broken.

Recovery Manager

Recovery Manager (RMAN) is now the preferred method of recovery from media failure.

Differences Between Release 7.2 and Release 7.3

Initialization Parameters

The following initialization parameters were added specifically for the Parallel Server Option:

- CLEANUP_ROLLBACK_ENTRIES
- DELAYED_LOGGING_BLOCK_CLEANOUTS
- GC_FREELIST_GROUPS
- GC_RELEASABLE_LOCKS

Data Dictionary Views

The following view was added specifically for the Parallel Server Option:

- FILE_LOCK

Dynamic Performance Views

The following view changed:

- V$BH

The following views were added:

- V$SORT_SEGMENT
- V$ACTIVE_INSTANCES

Free List Groups

You can now set free list groups for indexes, as well as for tables and clusters.

Fine Grain Locking

In Oracle Parallel Server release 7.3, PCM locks have additional options for configuration using fine grain locking. The changes affect the interpretation of the various parameters that determine the locks used to protect the database blocks in the distributed parallel server cache.

Fine grain locking is a more efficient method for providing locking in a multinode configuration. It provides a reduced rate of lock collision, and reduced space

requirements for managing locks, particularly in MPP systems. This feature relies on facilities provided by the hardware and operating system platform, and may not be available on all platforms.

Instance Registration

This feature enables each instance to register itself and certain of its attributes, and to establish contact with any other instance. Instance registration is transparent to the user, except in the case of parallel execution failure on remote instances of a parallel server. If a parallel query dies due to an error on a remote instance, the failed instance is now identified in the error message.

Sort Improvements

This release offers a more efficient way of allocating sort temporary space, which reduces serialization and cross-instance pinging. If you set up this capability correctly, it can particularly benefit Oracle Parallel Server performance in parallel mode.

For best results, try to establish stable sort space. Remember that sort space is cached in the instance. One instance does not release the space unless another instance runs out of space and issues a call to the first one to do so. This is an expensive, serialized process which hurts performance. If your system permanently deviates from stable sort space, it is better to over-allocate space, or simply not to use temporary tablespaces.

To determine the stability of your sort space, you can check the V$SORT_SEGMENT view. This new view shows every instance's history of sorting. If the FREED_EXTENTS and ADDED_EXTENTS columns show excessive allocation / deallocation activity, you should consider adding more space to the corresponding tablespace. Check also the FREE_REQUESTS value to determine if there is inter-instance conflict over sort space.

Another reason for excessive allocation and deallocation may be that some sorts are just too large. It may be worthwhile to assign a different temporary tablespace for the operations which require huge sorts. The MAX_SORT_SIZE value may help you to determine whether these large sorts have indeed occurred.

> **See Also:** *Oracle8i Designing and Tuning for Performance* for more information on sort enhancements.

XA Performance Improvements

Various scalability and throughput improvements have been made that affect XA transactions. These changes have no visible impact, other than improved performance.

The following three latches perform much better, and so enhance scalability:

- Global transaction mapping table latch
- Enqueues latch
- Session switching latch

Transaction throughput is enhanced because most of the common XA calls have reduced code path and reduced round-trips to the database.

XA Recovery Enhancements

Recovery of distributed transactions submitted through a TP monitor using the XA interface is now fully supported in Oracle Parallel Server.

The XA_RECOVER call has been enhanced, ensuring correct and complete recovery of one instance from transactions that have failed in another instance.

An option has been added to make the XA_RECOVER call wait for instance recovery. This feature enables one Oracle instance to do recovery on behalf of a failed Oracle instance, when both are part of the same Oracle Parallel Server cluster.

The XA_INFO string has a new clause called OPS_FAILOVER. If this is set to true for a given XA resource manager connection, any XA_RECOVER call issued from that connection will wait for any needed instance recovery to complete. The syntax is as follows:

OPS_FAILOVER=T

Upper- or lowercase (T or t) can be used. The default value of OPS_FAILOVER is false (F or f).

Previously, there was no guarantee that an XA_RECOVER call would return the list of in-doubt transactions from the failed instance. Setting OPS_FAILOVER=T ensures that this will happen.

When OPS_FAILOVER is set to true, the XA_RECOVER call will wait until SMON has finished cache recovery, has identified the in-doubt transactions, and added them to the PENDING_TRANS$ table that has a list of in-doubt transactions.

Deferred Transaction Recovery

Transaction recovery behavior has changed to allow:

- Greater database availability during startup
- Transactions to be recovered in parallel, if needed
- Recovery of long transactions without interfering with recovery of short transactions

Fast Warmstart

In previous releases, the database could not be opened until complete transaction recovery was performed after a failure. As of release 7.3, the database is opened for connections as soon as cache recovery is completed. (This only applies when opening the database, as opposed to doing failover in an Oracle Parallel Server environment.) In case of an instance failure, the database is available for connections through other running instances.

This means that active transactions as of the time of the failure are not yet rolled back; they appear active (holding row locks) to users of the system. Furthermore, all transactions system-wide that were active as of the time of failure are marked DEAD and the rollback segments containing these transactions are marked PARTIALLY AVAILABLE. These transactions are recovered as part of SMON recovery in the background, or by foreground processes that may encounter them, as described in the next section. The rollback segment is available for onlining.

Transaction Recovery

Given fast warmstart capability, the time needed to recover all transactions does not limit the general availability of the database. All data except the part locked by unrecovered transactions is now available to users. Given an OLTP workload, however, all the requests that were active when the database or instance went down will probably be resubmitted immediately. They will very likely encounter the locks held by the unrecovered transactions. The time needed to recover these transactions is thus still critical for access to the locked data. To alleviate this problem, transactions can now be recovered in parallel, if needed. Recovery can be done by the following operations.

Recovery by Foreground Processes. Rows may be locked by a transaction that has not yet been recovered. Any foreground process that encounters such a row can itself recover the transaction. The current recovery by SMON will still happen--so the entire transaction recovery will complete eventually. But if any foreground process runs into a row lock, it can quickly recover the transaction holding the lock,

and continue. In this way recovery operations are parallelized on a need basis: dead transactions will not hold up active transactions. Previously, active transactions had to wait for SMON to recover the dead transactions.

Recovery is done on a per-rollback segment basis. This prevents multiple foreground processes in different instances from recovering transactions in the same rollback segment, which would cause pinging. The foreground process fully recovers the transaction that it would otherwise have waited for. In addition, it makes a pass over the entire rollback segment and partially recovers all unrecovered transactions. It applies a configurable number of changes (undo records) to each transaction. This allows short transactions to be recovered quickly; without waiting for long transactions to be recovered. The initialization parameter CLEANUP_ROLLBACK_ENTRIES specifies the number of changes to apply.

Recovery by SMON. SMON transaction recovery operations are mostly unchanged. SMON is responsible for recovering transactions marked DEAD within its instance, transaction recovery during startup, and instance recovery. The only change is that it will make multiple passes over all the transactions that need recovery and apply only the specified number of undo records per transaction per pass. This prevents short transactions from waiting for recovery of a long transaction.

Recovery by Onlining Rollback Segment. Onlining a rollback segment now causes complete recovery of all transactions it contains. Previously, the onlining process posted SMON to do the recovery. Note that implicit onlining of rollback segments as part of warmstart or instance startup does not recover all transactions but instead marks them DEAD.

Load Balancing at Connect

In standard Oracle, load balancing now allows multiple listeners and multiple instances to be balanced at SQL*Net connect time. Multiple listeners can now listen on one Oracle instance, and the Oracle dispatcher will register with multiple listeners. The SQL*Net client layer will randomize multiple listeners via the DESCRIPTION_LIST feature.

For more information about load balancing at connect, please see the SQL*Net documentation for Oracle7 Server release 7.3.

Bypassing Cache for Sort Operations

The default value for the SORT_DIRECT_WRITES initialization parameter is now AUTO; it will turn itself on if your sort area is a certain size or greater. This will

improve performance. For more information, see the *Oracle8i Designing and Tuning for Performance.*

Delayed-Logging Block Cleanout

In Oracle7 Server release 7.3, the performance of delayed block cleanout is improved and related pinging is reduced. These enhancements are particularly beneficial for the Oracle Parallel Server.

Oracle7 Server release 7.3 provides a new initialization parameter, DELAYED_ LOGGING_BLOCK_CLEANOUTS, which is TRUE by default.

When Oracle commits a transaction, each block that the transaction changed is not immediately marked with the commit time. This is done later, upon demand--when the block is read or updated. This is called *block cleanout*. When block cleanout is done during an update to a current block, the cleanout changes and the redo records of the update are piggybacked with those of the update. In previous releases, when block cleanout was needed during a read to a current block, extra cleanout redo records were generated and the block was dirtied. This has been changed.

As of release 7.3, when a transaction commits, all blocks in the cache changed by the transaction are cleaned out immediately. This cleanout performed at commit time is a "fast version" which does not generate redo log records and does not repin the block. Most blocks will be cleaned out in this way, with the exception of blocks changed by long running transactions.

During queries, therefore, the data block's transaction information is normally up-to-date and the frequency with which block cleanout is needed is much reduced. Regular block cleanouts are still needed when querying a block where the transactions are still truly active, or when querying a block which was not cleaned out during commit.

During changes (INSERT, DELETE, UPDATE), the cleanout redo log records are generated and piggyback with the redo of the changes.

Parallel Execution Processor Affinity

Oracle7 Server release 7.3 provides improved defaults in the method by which servers are allocated among instances for the parallel execution option. As a result, users can now specify parallelism without giving any hints.

Parallel execution slaves are now assigned based on disk transfer rates and CPU processing rates for user queries. Work is assigned to query slaves that have preferred access to local disks versus remote disks, which is more costly. In this way data locality will improve parallel execution performance.

For best results, you should evenly divide data among the parallel server instances and nodes--particularly for moderate to large size tables that substantially dominate the processing. Data should be fairly evenly distributed on various disks, or across all the nodes. For very small tables, this is not necessary.

For example, if you have two nodes, a table should not be divided in an unbalanced way such that 90% resides on one node and 10% on the other node. Similarly, if you have four disks, one should not contain 90% of the data and the others contain only 10%. Rather, data should be spread evenly across available nodes and disks. This happens automatically if you use disk striping. If you do not use disk striping, you must manually ensure that this happens, if you desire optimum performance.

Differences Between Release 7.1 and Release 7.2

Pre-Allocating Space Unnecessary

For most parallel server configurations it is no longer necessary to pre-allocate data blocks to retain partitioning of data across free list groups. When a row is inserted, a group of data blocks is allocated to the appropriate free list group for an instance.

Data Dictionary Views

The following views were added specifically for the Parallel Server Option:

- FILE_LOCK
- FILE_PING

Dynamic Performance Views

The following views changed:

- V$BH
- V$CACHE
- V$PING
- V$LOCK_ACTIVITY

The following views were added:

- V$FALSE_PING
- V$LOCKS_WITH_COLLISIONS
- V$LOCK_ELEMENT

Free List Groups

It is now possible to specify a particular instance, and hence the free list group, from a session, using the command:

```
ALTER SESSION SET INSTANCE = instance_number
```

Table Locks

It is now possible to disable the ability for a user to lock a table using the command:

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

Re-enabling table locks is accomplished using the following command:

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

Lock Processes

The PCM locks held by a failing instance are now recovered by the lock processes of the instance recovering for the failed instance.

Differences Between Release 7.0 and Release 7.1

Initialization Parameters

- CACHE_SIZE_THRESHOLD was added.

Dynamic Performance Views

The following views changed:

- V$BH
- V$CACHE
- V$PING
- V$LOCK_ACTIVITY

Differences Between Version 6 and Release 7.0

This section describes differences between Oracle Version 6 and Oracle7 Release 7.0.

Version Compatibility

The Parallel Server Option for Version 6 is upwardly compatible with Oracle7 with one exception. In Version 6 all instances share the same set of redo log files, whereas in Oracle7 each instance has its own set of redo log files. *Oracle8i Migration* gives full details of migrating to Oracle7. After a database is upgraded to work with Oracle7 it cannot be started using a Oracle Version 6 server. Applications that run on Oracle7 may not run on Oracle Version 6.

File Operations

While the database is mounted in parallel mode, Oracle7 supports the following file operations that Oracle Version 6 only supported in exclusive mode:

- Adding, renaming, or dropping a data file
- Taking a data file offline or online
- Creating, altering, or dropping a tablespace
- Taking a tablespace offline or online

The instance that executes these operations may have the database open, as well as mounted.

Table A–1 shows the file operations and corresponding SQL statements that cannot be performed in Oracle Version 6 with the database mounted in parallel mode.

Table A–1 SQL Statements Now Supported in Oracle7

Operation	SQL statement
Creating a tablespace	CREATE TABLESPACE tablespace_name
Dropping a tablespace	DROP TABLESPACE tablespace_name
Taking a tablespace offline or online	ALTER TABLESPACE tablespace OFFLINE ALTER TABLESPACE tablespace ONLINE
Adding a data file	ALTER TABLESPACE tablespace ADD DATAFILE
Renaming a data file	ALTER TABLESPACE tablespace RENAME DATAFILE
Renaming a data file log file	ALTER TABLESPACE tablespace RENAME FILE
Adding a redo log file	ALTER DATABASE dbname ADD LOGFILE
Dropping a redo log file	ALTER DATABASE dbname DROP LOGFILE
Taking a data file offline or online	ALTER DATABASE dbname DATAFILE OFFLINE ALTER DATABASE dbname DATAFILE ONLINE

Oracle7 allows all of the file operations listed above while the database is mounted in shared mode.

A redo log file cannot be dropped when it is active, or when dropping it would reduce the number of groups for that thread below two. When taking a data file online or offline in Oracle7, the instance can have the database either open or closed and mounted. If any other instance has the database open, the instance taking the file online or offline must also have the database open.

Note: Whenever you add a data file, create a tablespace, or drop a tablespace and its data files, you should adjust the values of GC_FILES_TO_LOCKS and GC_DB_LOCKS, if necessary, before restarting Oracle in parallel mode. Failure to do so may result in an insufficient number of locks to cover the new file.

Deferred Rollback Segments

The global constant parameter GC_SAVE_ROLLBACK_LOCKS reserves distributed locks for deferred rollback segments, which contain rollback entries for transactions in tablespaces that were taken offline.

Version 6 does not support taking tablespaces offline in parallel mode, so the initialization parameter GC_SAVE_ROLLBACK_LOCKS is not necessary in Oracle Version 6. In Oracle7, this parameter is required for deferred rollback segments.

Redo Logs

In Oracle Version 6, all instances share the same set of online redo log files and each instance writes to the space allocated to it within the current redo log file.

In Oracle7, each instance has its own set of redo log files. A set of redo log files is called a thread of redo. Thread numbers are associated with redo log files when the files are added to the database, and each instance acquires a thread number when it starts up.

Log switches are performed on a per-instance basis in Oracle7; log switches in Oracle Version 6 apply to all instances, because the instances share redo log files.

Oracle7 introduces mirroring of online redo log files. The degree of mirroring is determined on a per-instance basis. This allows you to specify mirroring according to the requirements of the applications that run on each instance.

ALTER SYSTEM SWITCH LOGFILE

In Oracle Version 6, all instances shared one set of online redo log files. Therefore, the ALTER SYSTEM SWITCH LOGFILE statement forced all instances to do a log switch to the new redo log file.

There is no global option for this SQL statement in Oracle7, but you can force all instances to switch log files (and archive all online log files up to the switch) by using the ALTER SYSTEM ARCHIVE LOG CURRENT statement.

Initialization Parameters

The LOG_ALLOCATION parameter of Oracle Version 6 is obsolete in Oracle7. Oracle7 includes the new initialization parameter THREAD, which associates a set of redo log files with a particular instance at startup.

Free Space Lists

This section describes changes concerning free space lists.

Space Freed by Deletions and Updates

In Oracle Version 6, blocks freed by deletions or by updates that shrank rows are added to the common pool of free space. In Oracle7, blocks will go to the free list and free list group of the process that deletes them.

Free Lists for Clusters

In Oracle Version 6, the FREELISTS and FREELIST GROUPS storage options are not available for the CREATE CLUSTER statement, and the ALLOCATE EXTENT clause is not available for the ALTER CLUSTER statement.

In Oracle7, clusters (except for most hash clusters) can use multiple free lists by specifying the FREELISTS and FREELIST GROUPS storage options of CREATE CLUSTER and by assigning extents to instances with the statement ALTER CLUSTER ALLOCATE EXTENT (INSTANCE *n*).

Hash clusters in Oracle7 can have free lists and free list groups if they are created with a user-defined key for the hashing function and the key is partitioned by instance.

Initialization Parameters

The FREELISTS and FREELIST GROUPS storage options replace the initialization parameters FREE_LIST_INST and FREE_LIST_PROC of Oracle Version 6.

Import/Export

In Oracle Version 6, Export did not export free list information. In Oracle7, Export and Import can handle FREELISTS and FREELIST GROUPS.

SQL*DBA

STARTUP and SHUTDOWN must be done while disconnected in Version 6. In Oracle7, Release 7.0, STARTUP and SHUTDOWN must be issued while connected as INTERNAL, or as SYSDBA or SYSOPER.

In Oracle7, operations can be performed using either commands or the SQL*DBA menu interface, as described in *Oracle8i Utilities*.

Initialization Parameters

This section lists new parameters and obsolete parameters.

New Parameters

The new initialization parameter THREAD associates a set of redo log files with a particular instance at startup.

For a complete list of new parameters, refer to the *Oracle8i Reference.*

Obsolete Parameters

The following initialization parameters used in earlier versions of the Parallel Server Option are now obsolete in Oracle7.

- ENQUEUE_DEBUG_MULTI_INSTANCE
- FREE_LIST_INST
- FREE_LIST_PROC
- GC_SORT_LOCKS
- INSTANCES
- LANGUAGE
- LOG_ALLOCATION
- LOG_DEBUG_MULTI_INSTANCE
- MI_BG_PROCS (renamed to GC_LCK_PROCS)
- ROW_CACHE_ENQUEUE
- ROW_CACHE_MULTI_INSTANCE

For a complete list of obsolete parameters, refer to *Oracle8i Migration.*

Archiving

In Oracle Version 6, each instance archives the online redo log files for the entire parallel server because all instances share the same redo log files. You can therefore have the instance with easiest access to the storage medium use automatic archiving, while other instances archive manually.

In Oracle7, each instance has its own set of online redo log files so that automatic archiving only archives for the current instance. Oracle7 can also archive closed

threads. Manual archiving allows you to archive online redo log files for all instances. You can use the THREAD option of the ALTER SYSTEM ARCHIVE LOG statement to archive redo log files for any specific instance.

In Oracle7, the filenames of archived redo log files can include the thread number and log sequence number.

A new initialization parameter, LOG_ARCHIVE_FORMAT, specifies the format for the archived filename. A new database parameter, MAXLOGHISTORY, in the CREATE DATABASE statement can be specified to keep an archive history in the control file.

Media Recovery

Online recovery from media failure is supported in Oracle7 while the database is mounted in either parallel or exclusive mode.

In either mode, the database or object being recovered cannot be in use during recovery:

- To recover an entire database, it must be mounted but not open.
- To recover a tablespace, the database must be open and the tablespace must be offline.
- To recover data files (other than files in the SYSTEM tablespace), the database must be closed or open with the data files offline.

B

Restrictions

This appendix documents Oracle Parallel Server compatibility issues and restrictions. Topics in this appendix include:

- Compatibility Between Shared and Exclusive Mode
- Restrictions

Compatibility Between Shared and Exclusive Mode

The following sections describe aspects of compatibility between shared and exclusive modes on a parallel server:

The Export and Import Utilities

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

> **See Also:** *Oracle8i Utilities* for more information about Import and Export.

Compatibility Between Shared and Exclusive Modes

Oracle Parallel Server runs with any Oracle database created in exclusive mode. Each instance must have its own set of redo logs.

Oracle in exclusive mode can access a database created or modified by Oracle Parallel Server.

If Oracle Parallel Server allocates free space to a specific instance, that space may not be available for inserts for a different instance in exclusive mode. Of course, all data in the allocated extents is always available.

Restrictions

The following sections describe restrictions:

Maximum Number of Blocks Allocated at a Time

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

Restrictions in Shared Mode

Oracle running multiple instances in shared mode supports all the functionality of Oracle in exclusive mode, except as noted under the following headings:

Restricted SQL Statements

In shared mode, the following operations are not supported:

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

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

Maximum Number of Data Files

The number of data files supported by Oracle is operating system specific. Within this limit, the maximum number allowed depends on the values used in the CREATE DATABASE command, which in turn is limited by the physical size of the control file. This limit is the same in shared mode as in exclusive mode, but the additional instances of Oracle Parallel Server restrict the maximum number of files more than a single-instance system. For more details, see *Oracle8 SQL Reference*, and your Oracle operating system specific documentation.

Sequence Number Generators

Oracle Parallel Server does not support CACHE ORDER combination of options for sequence number generators in shared mode. Sequences created with both of these options are ordered but not cached when running in a parallel server.

Free Lists with Import and Export Utilities

The Export utility does not preserve information about multiple free lists and free list groups. When you export data from multiple instances and then, from a single node, import it into a file, the data may not end up distributed across extents in exactly the same way it was initially. The meta-data of the table into which it is imported contains the free list and free list group information that is henceforth associated with the data blocks.

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

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

X