Oracle8*i* Parallel Server

Concepts and Administration

Release 8.1.5

February 1999

Part No. A67778-01

ORACLE

Oracle8*i* Parallel Server Concepts and Administration

Part No. A67778-01

Release 8.1.5

Primary Author: Mark Bauer.

Primary Contributors: Wilson Chan, Andrew Holdsworth, Anjo Kolk, Rita Moran, Graham Wood, and Michael Zoll.

Contributors: Christina Anonuevo, Lance Ashdown, Bill Bridge, Sandra Cheever, Carol Colrain, Mark Coyle, Sohan Demel, Connie Dialeris, Karl Dias, Anurag Gupta, Deepak Gupta, Mike Hartstein, Ken Jacobs, Ashok Joshi, Jonathan Klein, Jan Klokkers, Boris Klots, Tirthankar Lahiri, Bill Lee, Lefty Leverenz, Juan Loaiza, Sajjad Masud, Neil Macnaughton, Ravi Mirchandaney, 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, and Betty Wu.

Graphic Designer: Valarie Moore.

Send Us Your Comments

Oracle8*i* Parallel Server Concepts and Administration, Release 8.1.5

Part No. A67778-01

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 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 the Information Development department using any of the following:

- Electronic mail: infodev@us.oracle.com
- FAX: (650) 506-7228 Attn: Oracle Server Documentation
- Postal service:
  Oracle Corporation
  Server Documentation Manager
  500 Oracle Parkway
  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 Worldwide Support Center.

Preface

This manual describes the Oracle Parallel Server (OPS) and supplements the *Oracle8i Administrator's Guide* and *Oracle8i Concepts*.

This manual prepares you to successfully implement parallel processing by providing a thorough presentation of the concepts and procedures involved. Information in this manual applies to OPS as it runs on all operating systems.

> **Note:** *Oracle8i Parallel Server Concepts and Administration* contains information that describes the features and functionality of the Oracle8*i* and the Oracle8*i* Enterprise Edition products. Oracle8*i* and the Oracle8*i* Enterprise Edition have the same basic features. However, several advanced features are available only with the Oracle8*i* Enterprise Edition, and some of these are optional. For example, to use client application failover, you must have the Enterprise Edition and the Parallel Server Option.

For information about the differences between Oracle8*i* and the Oracle8*i* Enterprise Edition and the available features and options, please refer to *Getting to Know Oracle8i*.

Intended Audience

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

Structure

Part I: Parallel Processing Fundamentals

Chapter 1, "Parallel Processing and Parallel Databases"	This chapter introduces parallel processing and parallel database technologies that offer great advantages for online transaction processing and decision support applications.
Chapter 2, "Implementing Parallel Processing"	This chapter explains how to attain the goals of speedup and scaleup by effectively implementing parallel processing and parallel database technology.
Chapter 3, "Parallel Hardware Architecture"	This chapter describes the range of available hardware implementations that allow parallel processing, and surveys their advantages and disadvantages.

Part II: Oracle Parallel Server Concepts

Chapter 4, "How Oracle Implements Parallel Processing"	This chapter gives a high-level view of how OPS provides high performance parallel processing.
Chapter 5, "Oracle Instance Architecture for Oracle Parallel Server"	This chapter explains features of Oracle multiple instance architecture that differ from an Oracle server in exclusive mode.
Chapter 6, "Oracle Database Architecture for the Parallel Server"	This chapter describes features of Oracle database architecture that pertain to the multiple instances of OPS.
Chapter 7, "Overview of Locking Mechanisms"	This chapter provides an overview of internal OPS locking mechanisms.
Chapter 8, "Integrated Distributed Lock Manager"	This chapter explains the role of the Integrated Distributed Lock Manager in controlling access to resources in OPS.

Chapter 9, "Parallel Cache Management Instance Locks"	This chapter provides a conceptual overview of PCM locks. The planning and allocation of PCM locks is one of the most complex tasks facing the Oracle Parallel Server database administrator.
Chapter 10, "Non-PCM Instance Locks"	This chapter describes some of the most common non-PCM instance locks.
Chapter 11, "Space Management and Free List Groups"	This chapter explains space management concepts.
Chapter 12, "Application Analysis"	This chapter provides a conceptual framework for optimizing OPS application design.

Part III: OPS System Development Procedures

Chapter 13, "Designing Databases for Parallel Server"	This chapter prescribes a general methodology for designing systems optimized for OPS.
Chapter 14, "Creating a Database and Objects for Multiple Instances"	This chapter describes aspects of database creation that are specific to OPS.
Chapter 15, "Allocating PCM Instance Locks"	This chapter explains how to allocate PCM locks to datafiles by specifying values for parameters in the initialization file of an instance.
Chapter 16, "Ensuring IDLM Capacity for Resources and Locks"	This chapter explains how to reduce contention for shared resources and gain maximum performance from OPS by ensuring that adequate space is available in the Integrated Distributed Lock Manager for all the necessary locks and resources.
Chapter 17, "Using Free List Groups to Partition Data"	This chapter explains how to allocate free lists and free list groups to partition data. By doing this you can minimize contention for free space when using multiple instances.

Part IV: OPS System Maintenance Procedures

Chapter 18, "Administering Multiple Instances"	This chapter describes how to administer instances of OPS.
Chapter 19, "Tuning to Optimize Performance"	This chapter provides an overview of tuning issues.

Chapter 20, "Cache Fusion and Inter-instance Performance"	This chapter describes Cache Fusion in detail and explains how to monitor Cache Fusion and inter-instance performance.
Chapter 21, "Backing Up the Database"	This chapter explains how to protect your data by archiving the online redo log files and periodically backing up the datafiles, the control file for your database, and the parameter files for your instances.
Chapter 22, "Recovering the Database"	This chapter describes Oracle recovery features on a parallel server.
Chapter 23, "Migrating from a Single Instance to Parallel Server"	This chapter describes database conversion from a single instance Oracle database to a multi-instance Oracle database using the parallel server option.

Part V: Reference

Appendix A, "Differences Among Versions"	This appendix describes the differences between this release and previous releases of the Oracle that pertain to OPS.
Appendix B, "Restrictions"	This appendix lists restrictions for OPS.

Related Documents

Before reading this manual, you should have already read *Oracle8i Concepts* and the *Oracle8i 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
rowid	The substitution value must be an expression of datatype ROWID.	00000462.0001.0001
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.

Contents

Send Us Your Comments iii

Preface v

Part I Parallel Processing Fundamentals

1 Parallel Processing and Parallel Databases

What Is Parallel Processing? 1-1
Parallel Processing Defined 1-2
Problems of Parallel Processing 1-4
Characteristics of a Parallel System 1-4
Parallel Processing for SMPs and MPPs 1-5
Parallel Processing for Integrated Operations 1-5
What Is a Parallel Server? 1-6
What Are the Key Elements of Parallel Processing? 1-6
Speedup and Scaleup: the Goals of Parallel Processing 1-6
Synchronization: A Critical Success Factor 1-9
Locking 1-11
Messaging 1-11
What Are the Benefits of Parallel Processing? 1-12
Enhanced Throughput: Scaleup 1-12
Improved Response Time: Speedup 1-13
What Are the Benefits of Parallel Databases? 1-13
Higher Performance 1-13

High Availability 1-14
Greater Flexibility 1-14
More Users 1-14
Do You Need Parallel Server? 1-14
Single Instance with Exclusive Access 1-15
Multi-Instance Database Systems 1-16
Distributed Database Systems 1-17
Client-Server Systems 1-20
What Is the Role of Parallel Execution? 1-21

2 Implementing Parallel Processing

The Four Levels of Scalability 2-1
Scalability of Hardware and Network 2-2
Scalability of Operating System 2-5
Scalability of Database Management System 2-5
Scalability of Application 2-6
When Is Parallel Processing Advantageous? 2-7
Data Warehousing Applications 2-7
Applications Updating Different Data Blocks 2-7
Failover and High Availability 2-8
Summary 2-8
When Is Parallel Processing Not Advantageous? 2-9
Guidelines for Effective Partitioning 2-10
Overview 2-10
Vertical Partitioning 2-10
Horizontal Partitioning 2-12
Common Parallel Processing Misconceptions 2-12

3 Parallel Hardware Architecture

Overview 3-1
Parallel Processing Hardware Implementations 3-2
Application Profiles 3-2
Required Hardware and Operating System Software 3-3
High Speed Interconnect 3-3
Globally Accessible Disk or Shared Disk Subsystem 3-3

Shared Memory Systems 3-3
Shared Disk Systems 3-5
Shared Nothing Systems 3-6
Overview of Shared Nothing Systems 3-7
Massively Parallel Systems 3-7
Summary of Shared Nothing Systems 3-8
Shared Nothing /Shared Disk Combined Systems 3-9

Part II Oracle Parallel Server Concepts

4 How Oracle Implements Parallel Processing

Enabling and Disabling Parallel Server 4-1
Synchronization 4-3
Block Level Locking 4-3
Row Level Locking 4-4
Space Management 4-4
System Change Number 4-4
High Performance Features 4-5
Fast Commits, Group Commits, and Deferred Writes 4-5
Row Locking and Multiversion Read Consistency 4-6
Online Backup and Archiving 4-6
Cache Fusion 4-6
Sequence Number Generators 4-7
Lamport SCN Generation 4-7
Free Lists 4-8
Free List Groups 4-8
Disk Affinity 4-8
Job and Instance Affinity 4-9
Transparent Application Failover 4-11
Cache Coherency 4-12
Parallel Cache Management Issues 4-12
Non-PCM Cache Management Issues 4-16

5 Oracle Instance Architecture for Oracle Parallel Server

Overview 5-1
Characteristics of OPS Multi-instance Architecture 5-4
System Global Area 5-5
Background Processes 5-5
Foreground Lock Acquisition 5-6
Cache Fusion Processing and the Block Server Process 5-7
Configuration Guidelines for Oracle Parallel Server 5-9

6 Oracle Database Architecture for the Parallel Server

File Structures 6-1
Control Files 6-1
Datafiles 6-2
Redo Log Files 6-3
The Data Dictionary 6-6
The Sequence Generator 6-6
The CREATE SEQUENCE Statement 6-6
The CACHE Option 6-6
The ORDER Option 6-7
Rollback Segments 6-7
Rollback Segments in OPS 6-8
Parameters Controlling Rollback Segments 6-9
Public and Private Rollback Segments 6-9
How Instances Acquire Rollback Segments 6-10

7 Overview of Locking Mechanisms

Differentiating Oracle Locking Mechanisms 7-1
Overview 7-1
Local Locks 7-2
Instance Locks 7-4
The LCK Process 7-6
The LMON and LMD0 Processes 7-7
Cost of Locks 7-7
Oracle Lock Names 7-8

Lock Name Format 7-8
PCM Lock Names 7-9
Non-PCM Lock Names 7-10
Coordination of Locking Mechanisms by the IDLM 7-12
The IDLM Tracks Lock Modes 7-12
The Instance Maps Database Resources to IDLM Resources 7-13
How IDLM Locks and Instance Locks Relate 7-14
The IDLM Provides One Lock Per Instance on a Resource 7-16

8 Integrated Distributed Lock Manager

What Is the Integrated Distributed Lock Manager? 8-1
The IDLM Grants and Coordinates Resource Lock Requests 8-1
Lock Requests Are Queued 8-2
Asynchronous Traps (ASTs) Communicate Lock Request Status 8-2
Lock Requests Are Converted and Granted 8-3
IDLM Lock Modes: Resource Access Rights 8-6
IDLM Features 8-8
Distributed Architecture 8-8
Fault Tolerance 8-8
Lock Mastering 8-9
Deadlock Detection 8-9
Lamport SCN Generation 8-9
Group-owned Locks 8-9
Persistent Resources 8-10
Memory Requirements 8-10
Support for MTS and XA 8-10
Views to Monitor IDLM Statistics 8-11

9 Parallel Cache Management Instance Locks

PCM Locks and How They Work 9-1
What PCM Locks Are 9-2
Allocation and Release of PCM Locks 9-3
How PCM Locks Operate 9-4
Number of Blocks per PCM Lock 9-6
Pinging: Signaling the Need to Update 9-8

Partitioning to Avoid Pinging 9-9
Lock Mode and Buffer State 9-10
How Initialization Parameters Control Blocks and PCM Locks 9-13
GC_* Initialization Parameters 9-13
Handling Data Blocks 9-15
Two Methods of PCM Locking: Fixed and Releasable 9-15
IDLM Lock Elements and PCM Locks 9-15
Number of Blocks per PCM Lock 9-17
Fine Grain Locking: Locks for One or More Blocks 9-18
How Fine Grain Locking Works 9-19
Performance Effects of Releasable Locking 9-20
Applying Fine Grain and Hashed Locking to Different Files 9-21
How Oracle Assigns Locks to Blocks 9-22
File to Lock Mapping 9-22
Number of Locks per Block Class 9-23
Lock Element Number 9-24
Examples: Mapping Blocks to PCM Locks 9-24
Setting GC_FILES_ TO_LOCKS 9-24
More Sample Hashed Settings of GC_FILES_TO_LOCKS 9-27
Sample Fine Grain Setting of GC_FILES_TO_LOCKS 9-28

10 Non-PCM Instance Locks

Overview 10-1
Transaction Locks (TX) 10-3
Table Locks (TM) 10-3
System Change Number (SC) 10-4
Library Cache Locks (N[A-Z]) 10-4
Dictionary Cache Locks (Q[A-Z]) 10-5
Database Mount Lock (DM) 10-5

11 Space Management and Free List Groups

How Oracle Handles Free Space 11-1
Overview 11-2
Database Storage Structures 11-2
Structures for Managing Free Space 11-4

Example: Free List Groups 11-8
SQL Options for Managing Free Space 11-11
Managing Free Space on Multiple Instances 11-11
Partitioning Free Space into Multiple Free Lists 11-11
Partitioning Data with Free List Groups 11-12
How Free Lists and Free List Groups Are Assigned to Instances 11-13
Free Lists Associated with Instances, Users, and Locks 11-14
Associating Instances with Free Lists 11-14
Associating User Processes with Free Lists 11-15
Associating PCM Locks with Free Lists 11-15
Controlling Extent Allocation 11-17
Automatic Allocation of New Extents 11-17
Pre-allocation of New Extents 11-17
Moving the High Water Mark of a Segment 11-18

12 Application Analysis

How Detailed Must Your Analysis Be? 12-1
Understanding Your Application Profile 12-2
Analyzing Application Functions and Table Access Patterns 12-2
Read-only Tables 12-2
Random SELECT and UPDATE Tables 12-3
INSERT, UPDATE, or DELETE Tables 12-4
Planning the Implementation 12-5
Partitioning Guidelines 12-5
Overview 12-5
Application Partitioning 12-6
Data Partitioning 12-7

Part III Oracle Parallel Server Development Procedures

13 Designing Databases for Parallel Server

Overview 13-1
Case Study: From Initial Database Design to OPS 13-2
"Eddie Bean" Catalog Sales 13-2
Tables 13-3
Users 13-3
Application Profile 13-3
Analyze Access to Tables 13-4
Table Access Analysis Worksheet 13-4
Case Study: Table Access Analysis 13-8
Analyze Transaction Volume by Users 13-9
Transaction Volume Analysis Worksheet 13-9
Case Study: Transaction Volume Analysis 13-10
Partition Users and Data 13-13
Case Study: Initial Partitioning Plan 13-13
Case Study: Further Partitioning Plans 13-14
Partition Indexes 13-16
Implement Hashed or Fine Grain Locking 13-17
Implement and Tune Your Design 13-18

14 Creating a Database and Objects for Multiple Instances

Creating a Database for a Multi-instance Environment 14-1
Summary of Tasks 14-1
Setting Initialization Parameters for Database Creation 14-2
Database Creation and Start Up 14-3
Setting CREATE DATABASE Options 14-3
Creating Database Objects to Support Multiple Instances 14-5
Creating Additional Rollback Segments 14-5
Configuring the Online Redo Log for OPS 14-8
Providing Locks for Added Datafiles 14-10
Changing the Value of CREATE DATABASE Options 14-10

15 Allocating PCM Instance Locks

Planning the Use and Maintenance of PCM Locks 15-2
Planning and Maintaining Instance Locks 15-2
Key to Allocating PCM Locks 15-2
Examining Datafiles and Data Blocks 15-3
Using Worksheets to Analyze PCM Lock Needs 15-4
Mapping Fixed PCM Locks to Data Blocks 15-5
Partitioning PCM Locks Among Instances 15-6
Setting GC_FILES_TO_LOCKS: PCM Locks for Each Datafile 15-6
GC_FILES_TO_LOCKS Syntax 15-7
Fixed Lock Examples 15-8
Releasable Lock Example 15-9
Guidelines 15-9
Tips for Setting GC_FILES_TO_LOCKS 15-10
Providing Room for Growth 15-10
Checking for Valid Number of Locks 15-11
Checking for Valid Lock Assignments 15-11
Setting Tablespaces to Read-only 15-11
Checking File Validity 15-12
Adding Datafiles without Changing Parameter Values 15-12
Setting Other GC_* Parameters 15-12
Setting GC_RELEASABLE_ LOCKS 15-13
Setting GC_ROLLBACK_ LOCKS 15-13
Tuning PCM Locks 15-14
Detecting False Pinging 15-14
How Much Time Do PCM Lock Conversions Take? 15-16
Which Sessions Are Waiting for PCM Lock Conversions to Complete? 15-17
What Is the Total Number of PCM Locks and Resources Needed? 15-17

16 Ensuring IDLM Capacity for Resources and Locks

Overview 16-1
Planning IDLM Capacity 16-2
Avoiding Dynamic Allocation of Resources and Locks 16-2
Computing Lock and Resource Needs 16-2
Monitoring Resource Utilization 16-3

Calculating the Number of Non-PCM Resources 16-4
Adjusting Oracle Initialization Parameters 16-6
Minimizing Table Locks to Optimize Performance 16-6
Setting DML_LOCKS to Zero 16-7
Disabling Table Locks 16-7

17 Using Free List Groups to Partition Data

Overview 17-2
Deciding How to Partition Free Space for Database Objects 17-2
Database Object Characteristics 17-2
Free Space Worksheet 17-5
Setting FREELISTS and FREELIST GROUPS in the CREATE Statement 17-6
FREELISTS Option 17-6
FREELIST GROUPS Option 17-6
Creating Free Lists for Clusters 17-7
Creating Free Lists for Indexes 17-8
Associating Instances, Users, and Locks with Free List Groups 17-9
Associating Instances with Free List Groups 17-10
Associating User Processes with Free List Groups 17-10
Associating PCM Locks with Free List Groups 17-11
Pre-allocating Extents (Optional) 17-11
The ALLOCATE EXTENT Option 17-11
Setting MAXEXTENTS, MINEXTENTS, and INITIAL Parameters 17-13
Setting the INSTANCE_NUMBER Parameter 17-13
Examples of Extent Pre-allocation 17-14
Dynamically Allocating Extents 17-15
Translation of Block Database Address to Lock Name 17-15
!*blocks* with ALLOCATE EXTENT Syntax 17-15
Identifying and Deallocating Unused Space 17-16
How to Determine Unused Space 17-16
Deallocating Unused Space 17-16
Space Freed by Deletions or Updates 17-16

Part IV Oracle Parallel Server System Maintenance Procedures

18 Administering Multiple Instances

Overview 18-2
Oracle Parallel Server Management 18-2
Defining Multiple Instances with Parameter Files 18-3
Using a Common Parameter File for Multiple Instances 18-3
Using Individual Parameter Files for Multiple Instances 18-4
Embedding a Parameter File Using IFILE 18-5
Specifying a Non-default Parameter File with PFILE 18-8
Setting Initialization Parameters for Multiple Instances 18-8
GC_* Global Cache Parameters 18-9
Parameter Notes for Multiple Instances 18-10
Parameters that Must Be Identical on All Instances 18-11
Determining the Amount of Locks Needed and Setting LM_* Parameters 18-12
Creating Database Objects for Multiple Instances 18-12
Starting Instances 18-13
Enabling Parallel Server and Starting Instances 18-13
Starting with OPS Disabled 18-14
Starting in Shared Mode 18-15
Specifying Instances 18-17
Differentiating Between Current and Default Instance 18-17
How SQL Statements Apply to Instances 18-18
How Server Manager Commands Apply to Instances 18-18
The Cluster Manager 18-22
OPS Cluster Administration 18-22
Specifying Instance Groups 18-23
Using a Password File to Authenticate Users on Multiple Instances 18-26
Shutting Down Instances 18-26
Limiting Instances for Parallel Query 18-27
PARALLEL_SERVER_INSTANCES 18-28
Instance Registration and Client/Service Connections 18-28
How Clients Access Services 18-29
Configuring Client-to-service Connections 18-31
Database Instance Registration 18-31

Connect Time Failover 18-32
Client Load Balancing 18-32
Connection Load Balancing 18-32
Parallel Execution Load Balancing 18-33
Managed Standby and Standby Databases 18-33

19 Tuning to Optimize Performance

General Guidelines 19-1
Overview 19-2
Keep Statistics for All Instances 19-2
Statistics to Keep 19-2
Change One Parameter at a Time 19-3
Contention 19-3
Detecting Lock Conversions 19-3
Locating Lock Contention within Applications 19-4
Tuning for High Availability 19-7
Detection of Error 19-8
Recovery and Re-mastering of IDLM Locks 19-8
Recovery of Failed Instance 19-8

20 Cache Fusion and Inter-instance Performance

The Role of Cache Fusion in Resolving Cache Coherency Conflicts 20-2
How Cache Fusion Produces Consistent Read Blocks 20-2
Partitioning Data to Improve Write / write Conflict Resolution 20-4
Improved Scalability with Cache Fusion 20-4
Reduced Context Switches and CPU Utilization 20-5
Reduced CPU Utilization with User-mode IPCs 20-5
Reduced I / O for Block Pinging and Reduced X-to-S Lock Conversions 20-6
Consistent-read Block Transfers by way of High Speed Interconnects 20-6
The Interconnect and Interconnect Protocols for OPS 20-6
Influencing Interconnect Processing 20-6
Supported Interconnect Software 20-7
Performance Expectations 20-7
Cache Fusion Block Request Latencies 20-8
Monitoring Cache Fusion and Inter-instance Performance 20-9

Goals of Monitoring Cache Fusion and OPS Performance 20-9
Latency Statistics in OPS 20-9
Statistics for Monitoring OPS and Cache Fusion 20-11
Creating OPS Data Dictionary Views with CATPARR.SQL 20-12
Global Dynamic Performance Views 20-12
Analyzing Global Cache and Cache Fusion Statistics 20-14
Analyzing Global Lock Statistics 20-18
Analyzing IDLM Resource, Lock, Message, and Memory Resource Statistics 20-20
IDLM Message Statistics 20-23
Analyzing OPS I/O Statistics 20-26
Analyzing Lock Conversions by Type 20-29
Analyzing Latch, OPS, and IDLM-related Statistics 20-31
Using V$SYSTEM_EVENTS to Identify Performance Problems 20-34
Events in V$SYSTEM_EVENTS Specifically Related to OPS 20-34
General Observations 20-35

21 Backing Up the Database

Choosing a Backup Method 21-2
Archiving the Redo Log Files 21-2
Archiving Mode 21-3
Automatic or Manual Archiving 21-3
Archive File Format and Destination 21-5
Redo Log History in the Control File 21-6
Backing Up the Archive Logs 21-7
Checkpoints and Log Switches 21-9
Checkpoints 21-9
Forcing a Checkpoint 21-9
Forcing a Log Switch 21-10
Forcing a Log Switch on a Closed Thread 21-11
Backing Up the Database 21-12
Open and Closed Database Backups 21-12
Recovery Manager Backup Issues 21-13
Operating System Backup Issues 21-15

22 Recovering the Database

Overview 22-1
Recovery from Instance Failure 22-2
Single-node Failure 22-2
Multiple-node Failure 22-3
Fast-Start Checkpointing 22-3
Fast-Start Roll Back 22-3
Access to Datafiles for Instance Recovery 22-4
Freezing the Database for Instance Recovery 22-4
Phases of Oracle Instance Recovery 22-5
Recovery from Media Failure 22-6
Complete Media Recovery 22-7
Incomplete Media Recovery 22-8
Restoring and Recovering Redo Log Files 22-9
Disaster Recovery 22-10
Parallel Recovery 22-14
Parallel Recovery Using Recovery Manager 22-14
Parallel Recovery Using Operating System Utilities 22-15
Fast-start Parallel Rollback in OPS 22-16
Managed Standby and Standby Databases 22-16

23 Migrating from a Single Instance to Parallel Server

Overview 23-1
Deciding to Convert 23-2
Reasons to Convert 23-2
Reasons Not to Convert 23-2
Preparing to Convert 23-2
Hardware and Software Requirements 23-3
Converting the Application from Single- to Multi-instance 23-3
Administrative Issues 23-3
Converting the Database from Single- to Multi-instance 23-4
Troubleshooting the Conversion 23-9
Database Recovery After Conversion 23-9
Loss of Rollback Segment Tablespace 23-9
Inadvisable NFS Mounting of Parameter File 23-9

Part V Reference

Differences Between 8.0.4 and 8.1 ..... A-1
Cache Fusion Architecture Changes A-1
New Views A-2
Removal of GMS A-2
Parallel Transaction Recovery is now "Fast-Start Parallel Rollback" A-2
Changes to Instance Registration A-3
Listener Load Balancing A-3
Diagnostic Enhancements A-4
Oracle Parallel Server Management (OPSM) A-4
Parallel Server Installation and Database Configuration A-4
Instance Affinity for Jobs A-4
Obsolete Parameters A-5
Differences Between Release 8.0.3 and Release 8.0.4 ..... A-5
New Initialization Parameters A-5
Obsolete Initialization Parameters A-5
Obsolete Startup Parameters A-6
Dynamic Performance Views A-6
Group Membership Services A-6
Differences Between Release 7.3 and Release 8.0.3 ..... A-6
New Initialization Parameters A-6
Obsolete GC_* Parameters A-6
Changed GC_* Parameters A-7
Dynamic Performance Views A-7
Global Dynamic Performance Views A-7
Integrated Distributed Lock Manager A-8
Instance Groups A-8
Group Membership Services A-8
Fine Grain Locking A-8
Client-side Application Failover A-8
Recovery Manager A-8
Differences Between Release 7.2 and Release 7.3 ..... A-9
Initialization Parameters A-9
Data Dictionary Views A-9
Dynamic Performance Views A-9

Free List Groups A-9
Fine Grain Locking A-9
Instance Registration A-10
Sort Improvements A-10
XA Performance Improvements A-11
XA Recovery Enhancements A-11
Deferred Transaction Recovery A-12
Load Balancing at Connect A-13
Bypassing Cache for Sort Operations A-14
Delayed-Logging Block Cleanout A-14
Parallel Query Processor Affinity A-15
Differences Between Release 7.1 and Release 7.2 ..... A-15
Pre-allocating Space Unnecessary A-15
Data Dictionary Views A-15
Dynamic Performance Views A-16
Free List Groups A-16
Table Locks A-16
Lock Processes A-16
Differences Between Release 7.0 and Release 7.1 ..... A-17
Initialization Parameters A-17
Dynamic Performance Views A-17
Differences Between Version 6 and Release 7.0 ..... A-17
Version Compatibility A-17
File Operations A-17
Deferred Rollback Segments A-19
Redo Logs A-19
Free Space Lists A-20
SQL*DBA A-20
Initialization Parameters A-21
Archiving A-21
Media Recovery A-22
Compatibility B-1
The Export and Import Utilities B-1
Compatibility Between Shared and Exclusive Modes B-1
Restrictions B-2

Maximum Number of Blocks Allocated at a Time B-2
Restrictions in Shared Mode B-2

Part I

Parallel Processing Fundamentals

1

Parallel Processing and Parallel Databases

This chapter introduces parallel processing and parallel database technologies. Both offer great advantages for Online Transaction Processing (OLTP) and Decision Support Systems (DSS). The administrator's challenge is to selectively deploy these technologies to fully use their multiprocessing powers.

To do this successfully you must understand how multiprocessing works, what resources it requires, and when you can and cannot effectively apply it. This chapter answers the following questions:

- What Is Parallel Processing?
- What Is a Parallel Server?
- What Are the Key Elements of Parallel Processing?
- What Are the Benefits of Parallel Processing?
- What Are the Benefits of Parallel Databases?
- What Is the Role of Parallel Execution?
- Do You Need Parallel Server?

What Is Parallel Processing?

This section defines parallel processing and describes its use.

- Parallel Processing Defined
- Problems of Parallel Processing
- Characteristics of a Parallel System
- Parallel Processing for SMPs and MPPs

- Parallel Processing for Integrated Operations

Parallel Processing Defined

Parallel processing divides a large task into many smaller tasks and executes the smaller tasks concurrently on several nodes. As a result, the larger task completes more quickly.

> **Note:** A *node* is a separate processor, often on a separate machine. Multiple processors, however, can reside on a single machine.

Some tasks can be effectively divided and are good candidates for parallel processing. For example, in a bank with one teller, customers must form one line to be served. With two tellers, the task of waiting on customers can be effectively split between the two tellers so customers are served twice as fast. This is an instance where parallel processing is an effective solution.

Not all tasks can be effectively divided. Assume that for the previous example, the bank manager must approve all loan requests. In this case, parallel processing does not necessarily improve service. No matter how many tellers are available to process loans, all requests must form a single line for bank manager approval. No amount of parallel processing can overcome this built-in restriction.

The following figures contrast sequential processing of a single parallel query with parallel processing of the same query. Figure 1–1 illustrates sequential processing in which a query executes as a single task.

Figure 1–1 Sequential Processing of a Single Task

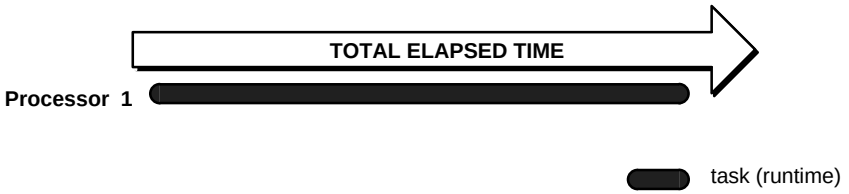
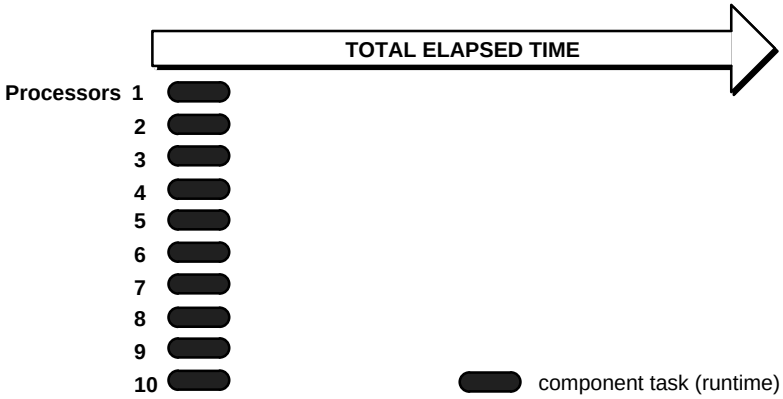


Figure 1–2 illustrates parallel processing in which a query is divided into multiple, smaller tasks, and each component task executes on a separate instance.

Figure 1–2 Parallel Processing: Executing Component Tasks in Parallel



These figures contrast sequential processing with parallel processing of multiple independent tasks from online transaction processing (OLTP) environments. Figure 1–3 shows sequential processing of multiple independent tasks.

Figure 1–3 Sequential Processing of Multiple Independent Tasks

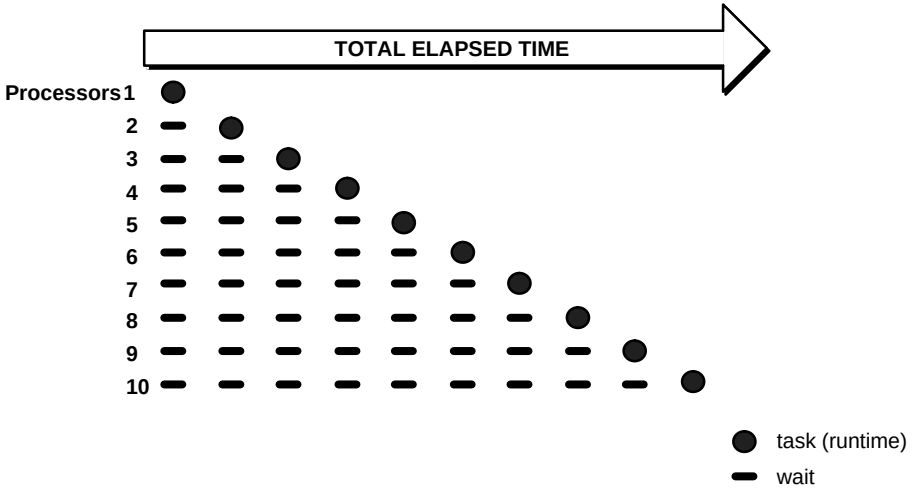
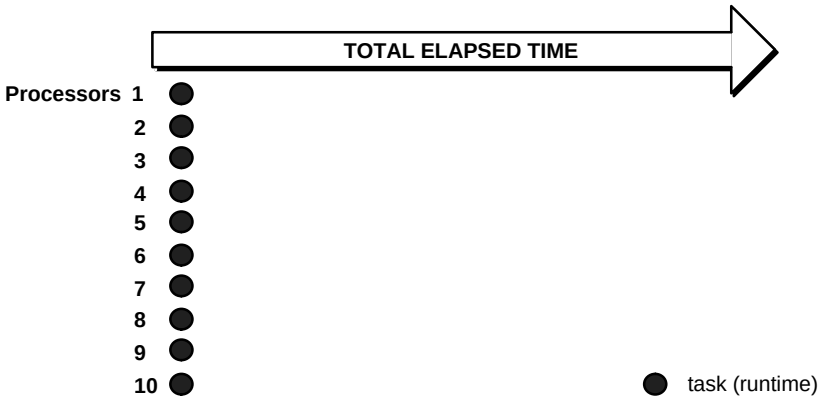


Figure 1–4 shows parallel processing of independent tasks.

Figure 1–4 Parallel Processing: Executing Independent Tasks in Parallel



In sequential processing, independent tasks compete for a single resource. Only task 1 runs without having to wait. Task 2 must wait until task 1 completes. Task 3 must wait until tasks 1 and 2 complete, and so on. Although the figure shows the independent tasks as the same size, the sizes of the tasks may vary.

By contrast, if you have a parallel server on a symmetric multiprocessor you can assign more CPU power to the tasks depending on how your CPUs are partitioned. Each independent task executes immediately on its own processor; no wait time is involved.

Problems of Parallel Processing

Effective implementation of parallel processing involves two challenges:

- Structuring tasks so some tasks execute at the same time "in parallel"
- Preserving task sequencing for tasks that must execute serially

Characteristics of a Parallel System

A parallel processing system has the following characteristics:

- Each processor in a system can perform tasks concurrently
- Tasks may need to be synchronized

- Nodes usually share resources, such as data, disks, and other devices

Parallel Processing for SMPs and MPPs

Parallel processing architectures support:

- Clustered and massively parallel processing (MPP) hardware where each node has its own memory
- Single memory systems, also known as "symmetric multiprocessing" (SMP) hardware, where multiple processors use one memory resource

Clustered and MPP machines have multiple memories, with each node typically having its own memory. Such systems promise significant price and performance benefits by using commodity memory and bus components to eliminate memory bottlenecks.

Database management systems supporting only one type of hardware limit the portability of applications, the potential to migrate applications to new hardware systems, and the scalability of applications. Oracle Parallel Server (OPS) exploits both clusters and MPP systems, and has no such limitations. Oracle without the Parallel Server Option exploits single CPU or SMP machines.

Parallel Processing for Integrated Operations

Parallel database software must effectively deploy the system's processing power to handle diverse applications such as online transaction processing (OLTP) applications, decision support system (DSS) applications, and mixtures of OLTP and DSS systems or "hybrid" systems.

OLTP applications are characterized by short transactions with low CPU and I/O usage. DSS applications have large transactions, with high CPU and I/O usage.

Parallel database software is often specialized, usually to serve as query processors. Since they are designed to serve a single function, however, specialized servers do not provide a common foundation for integrated operations. These include online decision support, batch reporting, data warehousing, OLTP, distributed operations, and high availability systems. Specialized servers have been used most successfully in the area of very large databases: in DSS applications, for example.

Versatile parallel database software should offer excellent price/performance on open systems hardware, and serve a wide variety of enterprise computing needs. Features such as online backup, data replication, portability, interoperability, and support for a wide variety of client tools can enable a parallel server to support application integration, distributed operations, and mixed application workloads.

What Is a Parallel Server?

A variety of hardware architectures allow multiple computers to share access to data, software, or peripheral devices. A parallel server is designed to take advantage of such architectures by running multiple instances that "share" a single physical database. In appropriate applications, a parallel server allows access to a single database by users on multiple machines with increased performance in terms of speedup and improved scaleup to process larger workloads.

A parallel server processes transactions in parallel by servicing a stream of transactions using multiple CPUs on different nodes where each CPU processes an entire transaction. Using parallel data manipulation language (PDML), one transaction can be executed by multiple nodes. This is an efficient approach because many applications consist of online insert and update transactions that tend to have short data access requirements. In addition to balancing the workload among CPUs, the parallel database provides concurrent access to data and ensures data integrity.

> **See Also:** "Do You Need Parallel Server?" on page 1-14 for a discussion of the Oracle configurations.

What Are the Key Elements of Parallel Processing?

This section describes key elements of parallel processing:

- Speedup and Scaleup: the Goals of Parallel Processing
- Synchronization: A Critical Success Factor
- Locking
- Messaging

Speedup and Scaleup: the Goals of Parallel Processing

You can measure the performance goals of parallel processing in terms of two important properties:

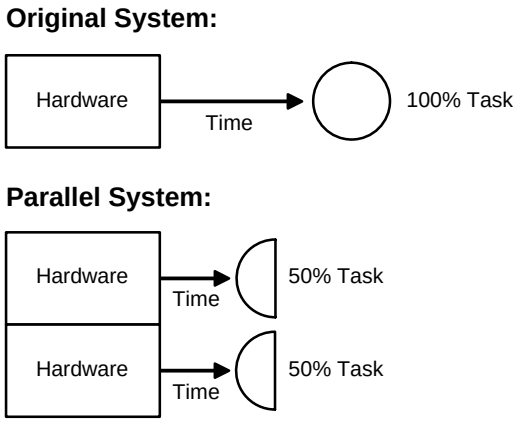
- Speedup
- Scaleup

Speedup

Speedup is the extent to which more hardware can perform the same task in less time than the original system. With added hardware, speedup holds the task

constant and measures time savings. Figure 1–5 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 1–5 Speedup



With good speedup, additional processors reduce system response time. You can measure speedup using this formula:

$$Speedup = \frac{Time\_Original}{Time\_Parallel}$$

Where:

Time_Original	is the elapsed time spent by a small system on the given task.
Time_Parallel	is the elapsed time spent by a larger, parallel system on the same task.

For example, if the original system took 60 seconds to perform a task, and two parallel systems took 30 seconds, then the value of speedup would equal 2.

$$2 = \frac{60}{30}$$

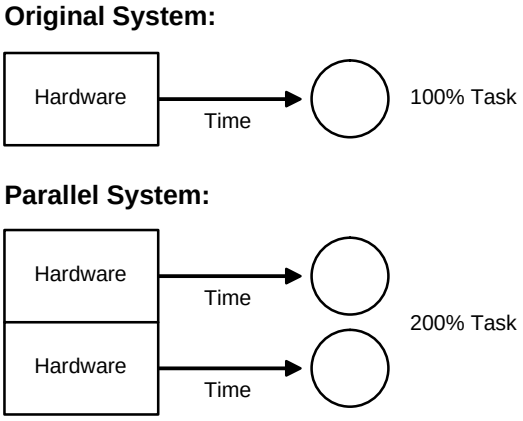
However, you may not experience direct, linear speedup. Instead, speedup may be more logarithmic. That is, assume the system can perform a task of size "x" in a time duration of "n". But for a task of size 2x, the system may require a time duration of 3n.

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

Scaleup

Scaleup 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 1–6 Scaleup



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.

Synchronization: A Critical Success Factor

Coordination of concurrent tasks is called *synchronization*. Synchronization is necessary for correctness. The key to successful parallel processing is to divide tasks so very little synchronization is necessary. The less synchronization necessary, the better the speedup and scaleup.

In parallel processing among nodes, having a high-speed interconnect among the parallel processors is helpful. The overhead of this synchronization can be very expensive if a great deal of inter-node communication is necessary. For parallel processing within a node, messaging is not necessary: shared memory is used instead. Messaging and locking between nodes is handled by the Integrated Distributed Lock Manager (IDLM).

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 significant synchronization may be necessary to coordinate many concurrent tasks.

Overhead

A great deal of time spent in synchronization indicates high contention for resources.

> **Note:** Too much time spent in synchronization can diminish the benefits of parallel processing. With less time spent in synchronization, better speedup and scaleup can be achieved.

Response time equals time spent waiting and time spent doing useful work. Table 1–1 illustrates how overhead increases as more concurrent processes are added. If 3 processes request a service at the same time, and they are served serially, then response time for process 1 is 1 second. Response time for process 2 is 2 seconds (waiting 1 second for process 1 to complete, then being serviced for 1 second). Response time for process 3 is 3 seconds (2 seconds waiting time plus 1 second service time).

Table 1–1 Increased Overhead with Increased Processes

Process Number	Service Time	Waiting Time	Response Time
1	1 second	0 seconds	1 second
2	1 second	1 second	2 seconds
3	1 second	2 seconds	3 seconds

One task, in fact, may require multiple messages. If tasks must continually wait to synchronize, then several messages may be needed per task.

Cost of Synchronization

While synchronization is a necessary element of parallel processing to preserve correctness, you need to manage its cost in terms of performance and system resources. Different types of parallel processing software may permit synchronization, but a given approach may or may not be cost-effective.

Sometimes you can achieve synchronization very inexpensively. In other cases the cost of synchronization may be too high. For example, if one table takes inserts from many nodes, a lot of synchronization is necessary. There may be high contention from the different nodes to insert into the same datablock: the datablock must be passed between the different nodes. This type of synchronization can be done, but the overhead in some cases might be significant.

See Also: Chapter 8, "Integrated Distributed Lock Manager", Chapter 12, "Application Analysis", and Chapter 19, "Tuning to Optimize Performance".

Locking

Locks are resource control mechanisms that synchronize tasks. Many different types of locking mechanisms are required to synchronize tasks required by parallel processing.

The Integrated Distributed Lock Manager (Integrated DLM, or IDLM) is the internal locking facility used with OPS. It coordinates resource sharing between nodes running a parallel server. The instances of a parallel server use the IDLM to communicate with each other and coordinate modification of database resources. Each node operates independently of other nodes, except when contending for the same resource.

The IDLM allows applications to synchronize access to resources such as data, software, and peripheral devices, so concurrent requests for the same resource are coordinated among applications running on different nodes.

The IDLM performs the following services for applications:

- Keeps track of the current "ownership" of a resource
- Accepts lock requests for resources from application processes
- Notifies the requesting process when a lock on a resource is available
- Notifies processes blocking a resource that they should release or downgrade a lock
- Obtains access to a resource for a process

See Also: Chapter 7, "Overview of Locking Mechanisms", for a discussion of locking mechanisms internal to the Oracle database, and Chapter 8, "Integrated Distributed Lock Manager".

Messaging

Parallel processing performs best when you have fast and efficient communication among nodes. The optimal type of system to have is one with high bandwidth and low latency that efficiently communicates with the IDLM. *Bandwidth* is the total size of messages that can be sent per second. *Latency* is the time in seconds that it takes to place a message on the interconnect and receive a response.

Most MPP systems and clusters have networks with reasonably high bandwidths. Latency, on the other hand, is an operating system issue that is mostly influenced by interconnect software and interconnect protocols. MPP systems, and most clusters, characteristically use interconnects with high bandwidth and low latency; other clusters may use Ethernet connections with relatively low bandwidth and high latency.

What Are the Benefits of Parallel Processing?

Parallel processing can benefit certain types of applications by providing:

- Enhanced Throughput: Scaleup
- Improved Response Time: Speedup

You can achieve improved response time either by breaking up a large task into smaller components or by reducing wait time, as shown in Figure 1–3.

Table 1–2 shows which types of workload can attain speedup and scaleup with properly implemented parallel processing.

Table 1–2 Speedup and Scaleup with Different Workloads

Workload	Speedup	Scaleup
OLTP	No	Yes
DSS	Yes	Yes
Batch (Mixed)	Possible	Yes
Parallel Query	Yes	Yes

Enhanced Throughput: Scaleup

If tasks can run independently of one another, they can be distributed to different CPUs or nodes and you can achieve scaleup: more processes can run through the database in the same amount of time.

If processes can run ten times faster, then the system can accomplish ten times more in the original amount of time. The parallel query 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. OPS without the parallel query 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 speedup by rewriting the batch programs and separating them into a number of parallel streams to take advantage of the multiple CPUs that are available.

Improved Response Time: Speedup

DSS applications and parallel query can attain speedup with parallel processing: each transaction can run faster. For OLTP applications, however, no speedup can be expected: only scaleup. With OLTP applications, each process is independent. Even with parallel processing, each insert or update on an order table still runs at the same speed. In fact, the overhead due to synchronization may cause a slight speed-down. Since each of the operations is small, it is inappropriate to attempt to parallelize them; the synchronization overhead would be greater than the benefit.

You can also achieve speedup with batch processing. The degree of speedup, however, depends on the cost and amount of synchronization between tasks.

What Are the Benefits of Parallel Databases?

Parallel database technology can benefit certain kinds of applications by enabling:

- Higher Performance
- High Availability
- Greater Flexibility
- More Users

Higher Performance

With more CPUs available to an application, higher speedup and scaleup can be attained. The performance improvement depends on the degree of inter-node locking and synchronization activities. Each lock operation is processor- and message-intensive; there can be a lot of latency. The volume of lock operations and database contention, as well as the throughput and performance of the IDLM, ultimately determine the scalability of the system.

High Availability

Nodes are isolated from each other, so a failure at one node does not bring the entire system down. One of the surviving nodes recovers the failed node and the system continues to provide data access to users. This means data is much more available than it would be with a single node upon node failure. This also amounts to significantly higher database availability.

Greater Flexibility

An OPS environment is extremely flexible. 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.

More Users

Parallel database technology can make it possible to overcome memory limits, enabling a single system to serve thousands of users.

Do You Need Parallel Server?

This section describes the following Oracle configurations that deliver high performance for different types of applications:

- Single Instance with Exclusive Access
- Multi-Instance Database Systems
- Distributed Database Systems
- Client-Server Systems

The parallel server is one of several Oracle options that provide high-performance relational databases serving many users. You can combine these configurations to suit your needs. A parallel server can be one of several servers in a distributed database environment, and the client-server configuration can combine various Oracle configurations into a hybrid system to meet specific application requirements.

Note: Support for any given Oracle configuration is platform-dependent; check to confirm that your platform supports the configuration you want.

For optimal performance, configure your system according to your particular application requirements and available resources, then design and tune the database and applications to make the best use of the configuration. Consider also the migration of existing hardware or software to the new system or to future systems.

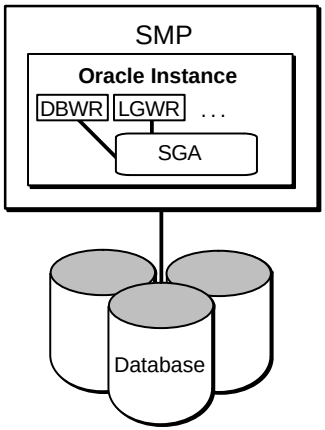
The following sections help you determine which Oracle configuration best meets your needs.

See Also: Chapter 3, "Parallel Hardware Architecture".

Single Instance with Exclusive Access

Figure 1–7 illustrates a single instance database system running on a symmetric multiprocessor (SMP). The database itself is located on a set of disks.

Figure 1–7 Single Instance Database System



A single instance accessing a single database can improve performance by running on a larger computer. A large single computer does not require coordination between several nodes and generally performs better than two small computers in a multinode system. However, two small computers often cost less than one large one.

The cost of redesigning and tuning your database and applications for OPS might be significant if you want to migrate from a single computer to a multi-node

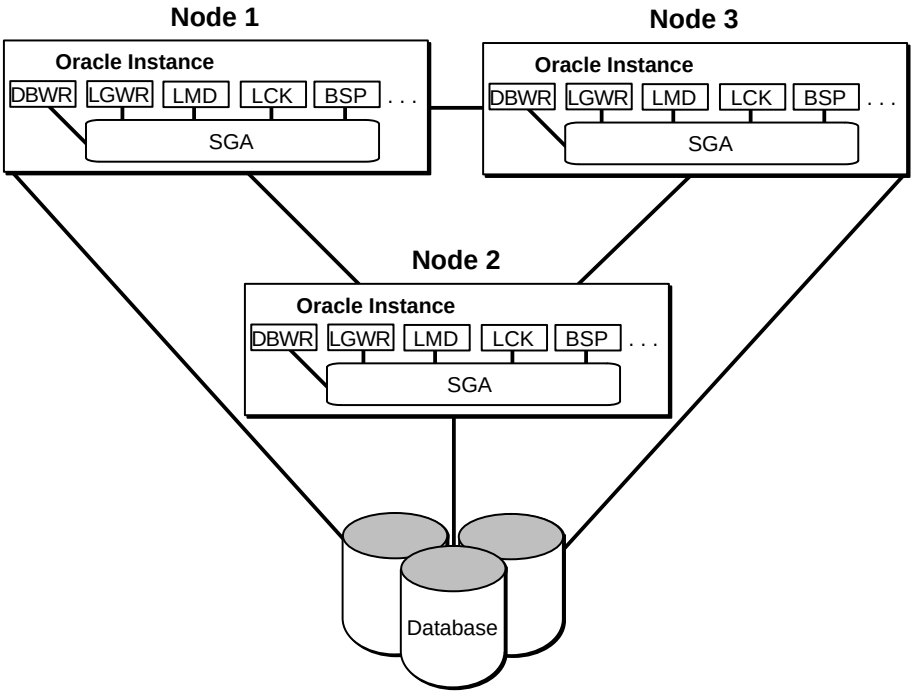
system. In situations like this, consider whether a larger, single computer might be a better solution than moving to a parallel server.

> **See Also:** *Oracle8i Concepts* for more information about single instance Oracle.

Multi-Instance Database Systems

Figure 1–8 illustrates the OPS option running on a cluster or MPP. This configuration is also referred to as a "multi-instance database system". OPS is an excellent solution for applications that can be configured to minimize the passing of data between instances on different nodes.

Figure 1–8 Multi-Instance Database System



This system requires the LMD, LCK, and BSP processes as well as foreground processes on each instance. These processes coordinate global locking by communicating directly from one instance to another by way of the interconnect.

Note: BSP is the Block Server Process that only exists in an OPS environment. BSP manages out-going messages to requesting nodes as well as the transmission of consistent read blocks from one node to another. The LMD process manages in-coming lock requests for instances holding locks needed by other instances.

See Also: For more information about BSP, please refer to "Cache Fusion Processing and the Block Server Process" on page 5-7 and Chapter 20, "Cache Fusion and Inter-instance Performance".

In OPS, instances are decoupled from databases. In exclusive mode, there is a one-to-one correspondence of instance to database. In shared (parallel) mode, however, there can be many instances to a single database.

In general, any single application performs best when it has exclusive access to a database on a larger system, as compared with its performance on a smaller node of a multinode environment. This is because the cost of synchronization automatically increases if you migrate to a multinode environment. The performance difference depends on characteristics of that application and all other applications sharing access to the database.

Applications with one or both of the following characteristics are well suited to run on separate instances of a parallel server:

- Applications that primarily only read (not update) data
- Applications that either change disjoint groups of datablocks or change the same datablocks at different times

See Also: "Enabling and Disabling Parallel Server" on page 4-1, Chapter 8, "Integrated Distributed Lock Manager", and *Oracle8i Concepts* for more information about the DBWR, LGWR, LMD, and LCK background processes.

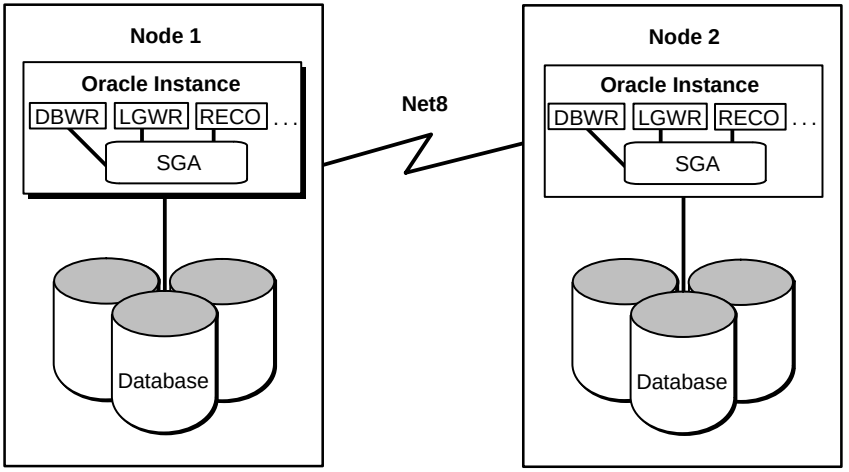
Distributed Database Systems

You can link several Oracle servers and databases to form a *distributed database* system. This configuration includes multiple databases, each of which is accessed directly by a single server and which can be accessed indirectly by other instances through server-to-server cooperation. You can use each node for database processing, but the data is permanently partitioned among the nodes. A parallel server, in contrast, has multiple instances with direct access to one database.

Note: Oracle Parallel Server can one component of a distributed database.

Figure 1–9 illustrates a distributed database system. This database system requires the RECO background process on each instance. There is no LCK, LMON, or LMD background process because this is not an OPS configuration, and the Integrated Distributed Lock Manager is not needed.

Figure 1–9 Distributed Database System

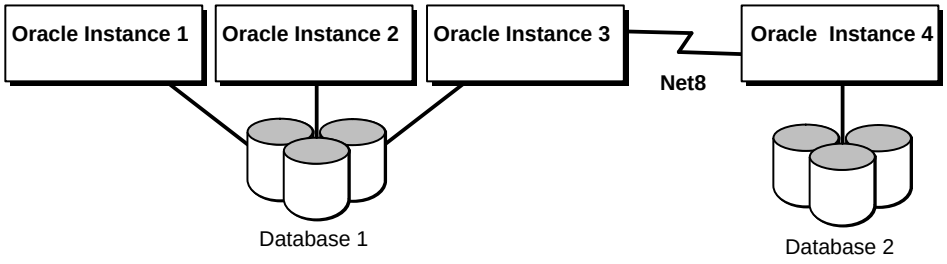


The multiple databases of a distributed system can be treated as one logical database, because servers can access remote databases transparently using Net8.

If you can partition your data into multiple databases with minimal overlap, you can use a distributed database system instead of a parallel server, sharing data between the databases, as mentioned, with Net8. A parallel server provides automatic data sharing among nodes through the common database.

A distributed database system allows data to reside at several widely separated sites. Users can access data from geographically separated databases providing network connections exist between the separate nodes. A parallel server requires all data to be at a single site because of the requirement for low latency, high bandwidth communication among nodes. But a parallel server can also be part of a distributed database system as illustrated in Figure 1–10.

Figure 1–10 Oracle Parallel Server as Part of a Distributed Database



Multiple databases require separate database administration, and a distributed database system requires coordinated administration of the databases and network protocols. A parallel server can consolidate several databases to simplify administrative tasks.

Multiple databases can provide greater availability than a single instance accessing a single database, because an instance failure in a distributed database system does not prevent access to data in the other databases: only the database owned by the failed instance is inaccessible. A parallel server, however, allows continued access to all data when one instance fails, including data accessed by the instance running on the failed node.

A parallel server accessing a single consolidated database avoids the need for distributed updates, inserts, or deletions and more expensive two-phase commits by allowing a transaction on any node to write to multiple tables simultaneously, regardless of which nodes usually write to those tables.

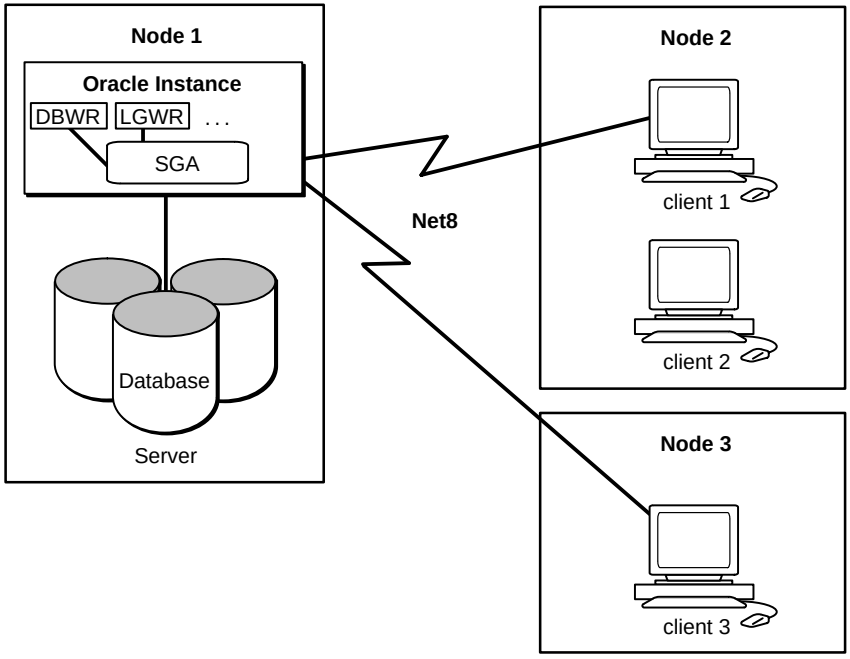
> **See Also:** *Oracle8i Backup and Recovery Guide* for more information about instance recovery and *Oracle8i Distributed Database Systems* for more information about Oracle distributed database features.

Client-Server Systems

Any Oracle configurations can run in a client-server environment. In Oracle, a client application runs on a remote computer using Net8 to access an Oracle server through a network. The performance of this configuration is typically limited to the power of the single server node.

Figure 1–11 illustrates an Oracle client-server system.

Figure 1–11 Client-Server System



Note: Client-server processing is suitable for any Oracle configuration. Check your Oracle platform-specific documentation to see whether it is implemented on your platform.

The client-server configuration allows you to off-load processing from the computer that runs an Oracle server. If you have too many applications running on one machine, you can off-load them to improve performance. However, if your database

server is reaching its processing limits you might want to move either to a larger machine or to a multinode system.

For compute-intensive applications, you could run some applications on one node of a multinode system while running Oracle and other applications on another node or on several other nodes. In this way you could use various nodes of a parallel machine as client nodes and one as a server node.

If the database has several distinct, high-throughput parts, a parallel server running on high-performance nodes can provide quick processing for each part of the database while also handling occasional access across parts.

A client-server configuration requires that the network convey all communications between clients and the database. This may not be appropriate for high-volume communications as is required for many batch applications.

See Also: "Client-Server Architecture" in *Oracle8i Concepts.*

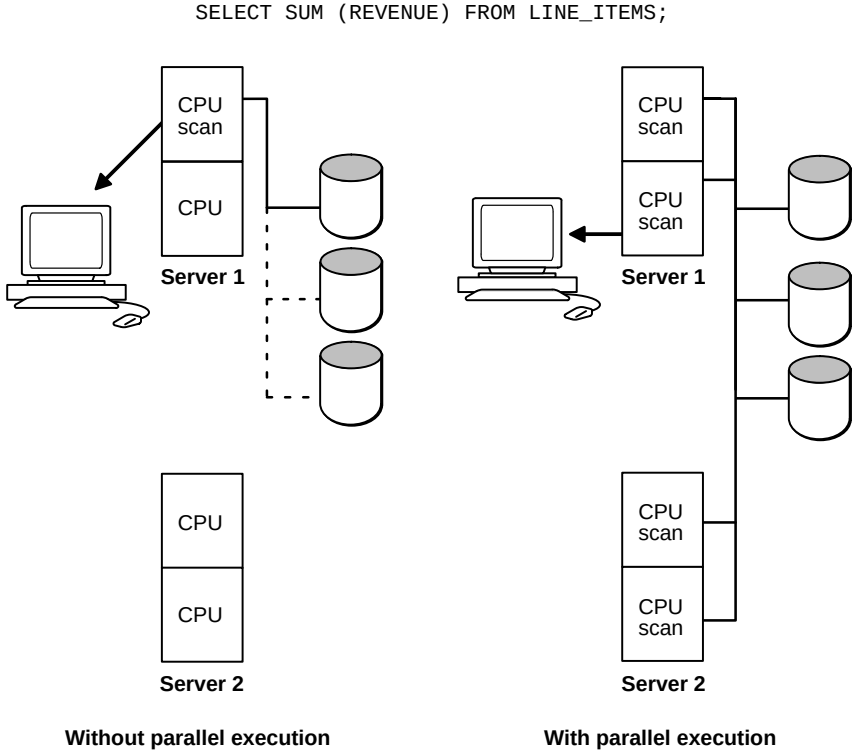
What Is the Role of Parallel Execution?

With its parallel execution features, Oracle can divide the work of processing certain types of SQL statements among multiple query server processes.

OPS provides the framework for parallel execution to work between nodes. Parallel execution features behave the same way in Oracle with or without the Parallel Server Option. The only difference is that OPS enables multiple nodes to execute on behalf of a single query or other parallel operation.

In some applications, notably data warehousing applications, individual queries consume a great deal of CPU resources and require significant disk I/O, unlike most online insert or update transactions. To take advantage of multiprocessing systems, the data server must parallelize individual queries into units of work that can be processed simultaneously. Figure 1–12 shows an example of parallel query processing.

Figure 1–12 Example of Parallel Execution Processing



If the query was not processed in parallel, disks would be read serially with a single I/O. A single CPU would have to scan all rows in the LINE_ITEMS table and total the revenues across all rows. With the query parallelized, disks are read in parallel, with multiple I/Os. Several CPUs can scan a part of the table in parallel and aggregate the results. Parallel query benefits not only from multiple CPUs but also from more of the available I/O bandwidth.

> **See Also:** *Oracle8i Concepts* and *Oracle8i Tuning* for detailed explanations of parallel execution.

2

Implementing Parallel Processing

There is an old network saying: Bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed—you can't bribe God.

— David Clark, MIT

To attain speedup and scaleup, you must effectively implement parallel processing and parallel database technology. This means designing and building your system for parallel processing from the start. This chapter covers the following issues:

- The Four Levels of Scalability
- When Is Parallel Processing Advantageous?
- When Is Parallel Processing Not Advantageous?
- Guidelines for Effective Partitioning
- Common Parallel Processing Misconceptions

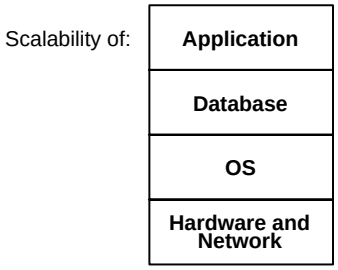
The Four Levels of Scalability

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.

Figure 2–1 Levels of Scalability



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.

See Also: "Required Hardware and Operating System Software" on page 3-3.

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 capacity for 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 query, rely on high bandwidth.

Disk Input and Output

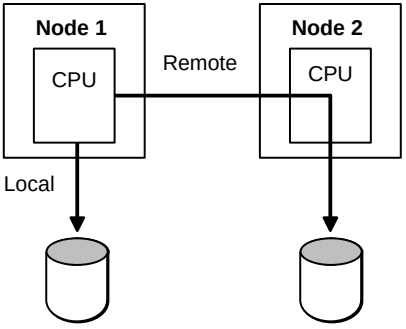
Local I/Os are faster than remote I/Os (those which occur between nodes). If a great deal of remote I/O is needed, the system loses scalability. In this case you can partition data so that the data is local. Figure 2–2 illustrates the difference.

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.

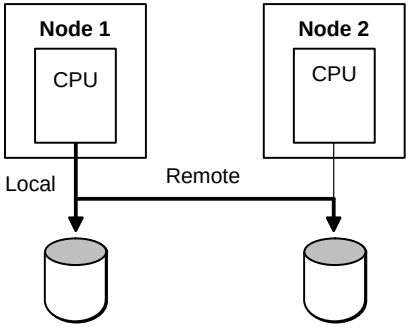
The shared disk architectures shown in Figure 2–2 are explained in the next chapter, Chapter 3, "Parallel Hardware Architecture".

Figure 2–2 Local and Remote I/O on Shared Nothing and Shared Disk

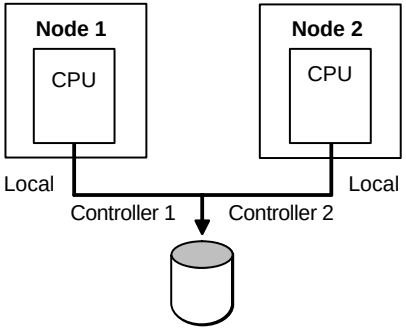
Shared Nothing



Shared Disk Cluster



Shared Disk Cluster with Dual Ported Controller

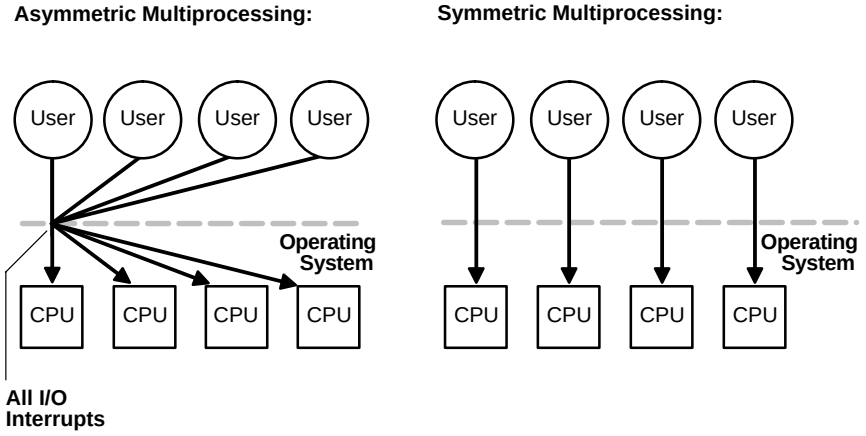


Scalability of Operating System

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:

Figure 2–3 Asymmetric Multiprocessing vs. Symmetric Multiprocessing



Here, potential scalability of the hardware is lost because the operating system can only process one resource request at a time. Each time a request enters the operating system, a lock is held to exclude the others. In symmetrical multiprocessing, by contrast, there is no such restriction.

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 speedup and scaleup.

> **See Also:** "Disk Affinity" on page 4-8, and the chapters on Parallel Execution in *Oracle8i Tuning*.

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 datablock. 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 the above 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:** Chapter 13, "Designing Databases for Parallel Server" and Chapter 12, "Application Analysis".

When Is Parallel Processing Advantageous?

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

- Data Warehousing Applications
- Applications Updating Different Data Blocks
- Failover and High Availability
- Summary

Data Warehousing Applications

Data warehousing applications that infrequently update, insert, or delete data are often appropriate for Oracle Parallel Server (OPS). 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 to be 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 OPS 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 Updating 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 fine grained locking further improve performance of such applications.

Departmentalized Applications

Applications that primarily modify different tables in the same database are also suitable for OPS. 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.

Failover and High Availability

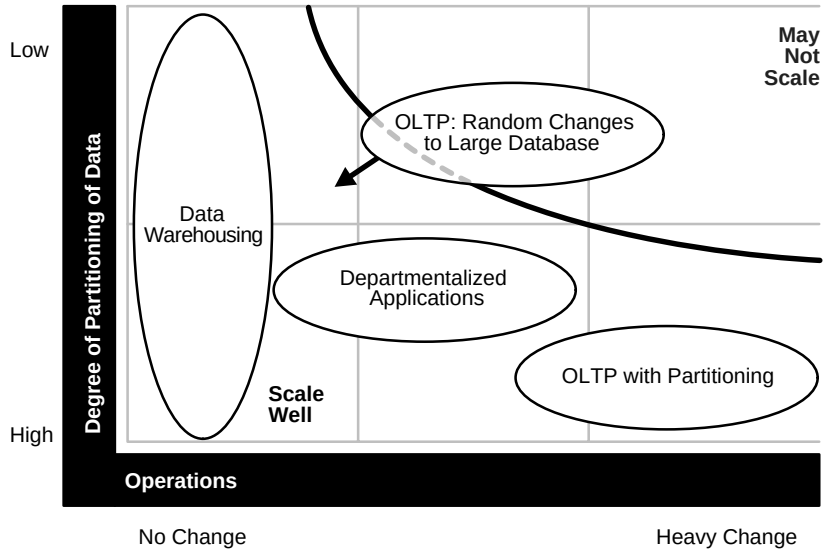
Applications requiring high availability benefit from the Oracle Parallel Server's failover capability. If the connection through one instance to the database is broken, you can write applications to automatically reconnect through a different instance.

Summary

Figure 2–4 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 making random changes to large databases were historically not considered good parallel server candidates. Such applications, however, are becoming more scalable with advanced intra-node communication by way of the 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 2–4 Scalability of Applications



When Is Parallel Processing Not Advantageous?

The following guidelines describe situations when parallel processing is *not* advantageous.

- In general, parallel processing is less advantageous when the cost of synchronization becomes too high and therefore throughput decreases.

  If many users on a large number of nodes modify a small set of data, then synchronization is likely to be very high. However, if they just read data, then no synchronization is required.

- Parallel processing is not advantageous when there is contention between instances on a single block or row.

  For example, it would not be effective to use a table with one row used primarily as a sequence numbering tool. Such a table would be a bottleneck because every process would have to select the row, update it, and release it sequentially.

Guidelines for Effective Partitioning

This section provides general guidelines for partitioning decisions that decrease synchronization and improve performance.

- Overview
- Vertical Partitioning
- Horizontal Partitioning

Overview

You can partition any of the three elements of processing, depending on function, location, and so on, such that they do not interfere with each other. These elements are:

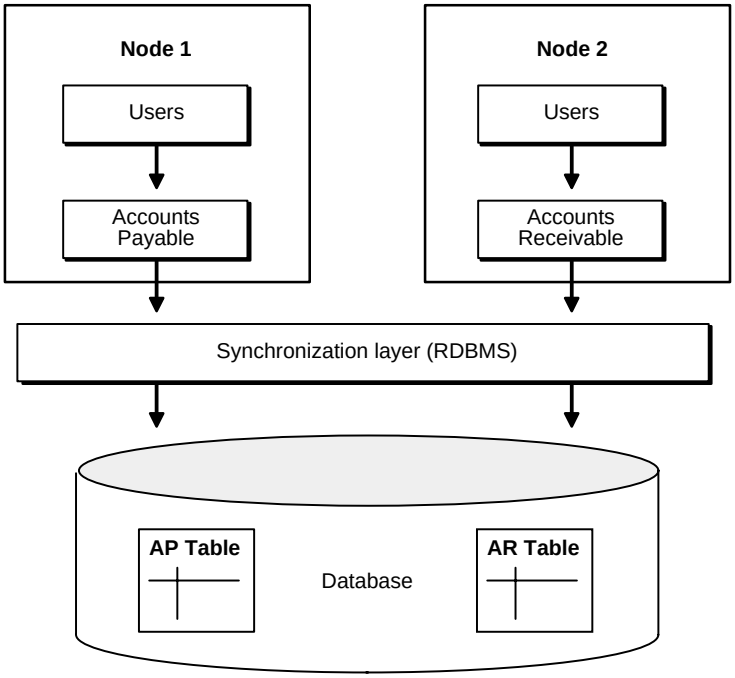
- Users
- Applications
- Data

Partition data, based on groups of users who access it; partition applications into groups that access the same data. Also consider geographic partitioning or partitioning by location.

Vertical Partitioning

With vertical partitioning, many tasks can run on a large number of resources without much synchronization. Figure 2–5 illustrates vertical partitioning.

Figure 2–5 Vertical Partitioning



Here, a company's accounts payable and accounts receivable functions have been partitioned by users, application, and data. They have been placed on two separate nodes. Here, most synchronization occurs on the same node; this is very efficient. The cost of synchronization on the local node is cheaper than the cost of synchronization *between* nodes.

Partition tasks on a subset of resources to reduce synchronization. When you partition, a smaller set of tasks will require access to shared resources.

Horizontal Partitioning

To illustrate the concept of horizontal partitioning, Figure 2–6 represents the rows of a stock table. If OPS has four instances, each on its own node, then partition them so that each instance accesses only a subset of the data.

Figure 2–6 Horizontal Partitioning

Node 1	Node 2	Node 3	Node 4
Rows	Rows	Rows	Rows
1 Through 10	11 Through 20	21 Through 30	31 Through 40

In this example, very little synchronization is necessary because the instances access different sets of rows. Similarly, users partitioned by location can often run almost independently. Very little synchronization is needed if users do not access the same data.

> **See Also:** For more information on partitioning, please refer to *Oracle8i Tuning*.

Common Parallel Processing Misconceptions

Various mistaken notions can lead to unrealistic expectations about parallel processing. Consider the following:

- Do not assume that when switching to parallel processing it will automatically work the way you expect. A good deal of database design and application tuning is required to make parallel processing successful.
- Scalability is not determined just by the number of nodes or CPUs involved, but also by the interconnect's bandwidth and latency, and by the amount and cost of synchronization.

  In some applications, a single synchronization (hotshot) may be so expensive as to constitute a problem; in other applications, many synchronizations on less contentious data may be perfectly acceptable.
- Just having parallel processing does not automatically mean higher availability: higher availability depends on your system architecture.

For example, on some MPP systems if one CPU dies, the entire machine dies. On a cluster, by contrast, if one node dies other nodes survive. The same is also true for MPP systems.

- All applications may not have been designed to scale effectively.

3

Parallel Hardware Architecture

You can deploy Oracle Parallel Server (OPS) on various architectures. This chapter describes hardware implementations that accommodate the parallel server and explains their advantages and disadvantages.

- Overview
- Required Hardware and Operating System Software
- Shared Memory Systems
- Shared Disk Systems
- Shared Nothing Systems
- Shared Nothing /Shared Disk Combined Systems

Overview

This section covers the following topics:

- Parallel Processing Hardware Implementations
- Application Profiles

Oracle configurations support parallel processing within a machine, between machines, and between nodes. There is no advantage to running OPS on a single node with a single instance; you would incur overhead without receiving benefits. With standard Oracle you do not have to do anything special on shared memory configurations to take advantage of some parallel processing capabilities.

Although this manual focuses on OPS on a shared nothing/shared disk architecture, the application design issues discussed in this book may also be relevant to standard Oracle systems.

Parallel Processing Hardware Implementations

We often categorize parallel processing hardware implementations according to the particular resources that are shared. This chapter describes these categories:

- Shared memory systems
- Shared disk systems
- Shared nothing systems

These implementations can also be described as "tightly coupled" or "loosely coupled", according to the way the nodes communicate.

Oracle supports *all* these implementations of parallel processing, assuming that in a shared nothing system the software enables a node to access a disk from another node. For example, the IBM SP2 features a virtual shared disk: the disk is shared through software.

Note: Support for any given Oracle configuration is platform-dependent; check whether your platform supports your desired configuration.

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.

Required Hardware and Operating System Software

Each hardware vendor implements parallel processing in its own way, but the following common elements are required for OPS:

- High Speed Interconnect
- Globally Accessible Disk or Shared Disk Subsystem

High Speed Interconnect

This is a high bandwidth, low latency communication facility among nodes for lock manager and cluster manager traffic. The interconnect can be Ethernet, FDDI, or some other proprietary interconnect method. If the primary interconnect fails, a back-up interconnect is usually available. The back-up interconnect ensures high availability, and prevents single points of failure.

Globally Accessible Disk or Shared Disk Subsystem

All nodes in loosely coupled or massively parallel systems have simultaneous access to shared disks. This gives multiple instances of Oracle8 concurrent access to the same database. These shared disk subsystems are most often implemented by way of shared SCSI or twin-tailed SCSI (common in UNIX) connections to a disk farm. On some MPP platforms, such as IBM SP, disks are associated to nodes and a virtual shared disk software layer enables global access to all nodes.

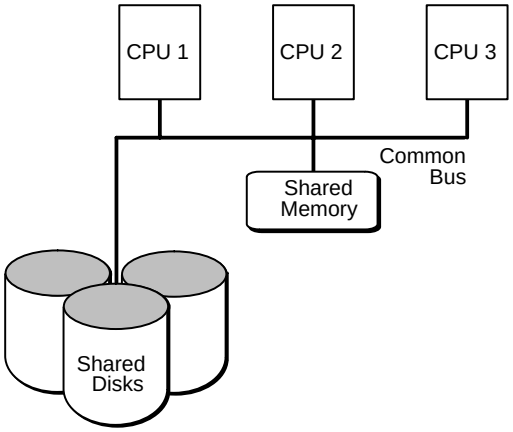
Note: The Integrated Distributed Lock Manager (IDLM) coordinates modifications of data blocks, maintenance of cache consistency, recovery of failed nodes, transaction locks, dictionary locks, and SCN locks.

Shared Memory Systems

Tightly coupled shared memory systems, illustrated in Figure 3–1, have the following characteristics:

- Multiple CPUs share memory
- Each CPU has full access to all shared memory through a common bus
- Communication among nodes occurs by way of shared memory
- Performance is limited by memory bus bandwidth

Figure 3–1 Tightly Coupled Shared Memory System



Symmetric multiprocessor (SMP) machines are often comprised of nodes in a cluster. You can install multiple SMP nodes with OPS in a tightly coupled system where memory is shared among the multiple CPUs, and is accessible by all the CPUs through a memory bus. Examples of tightly coupled systems include the Pyramid, Sequent, and Sun SparcServer.

It does not make sense to run OPS on a single SMP machine, because the system would incur a great deal of unnecessary overhead from IDLM accesses.

Performance is potentially limited in a tightly coupled system 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 common bus bandwidth.

Parallel processing advantages of shared memory systems are these:

- Memory access is less expensive than inter-node communication: this means internal synchronization is faster than using the Lock Manager
- Shared memory systems are easier to administer than a cluster

A disadvantage of shared memory systems for parallel processing is:

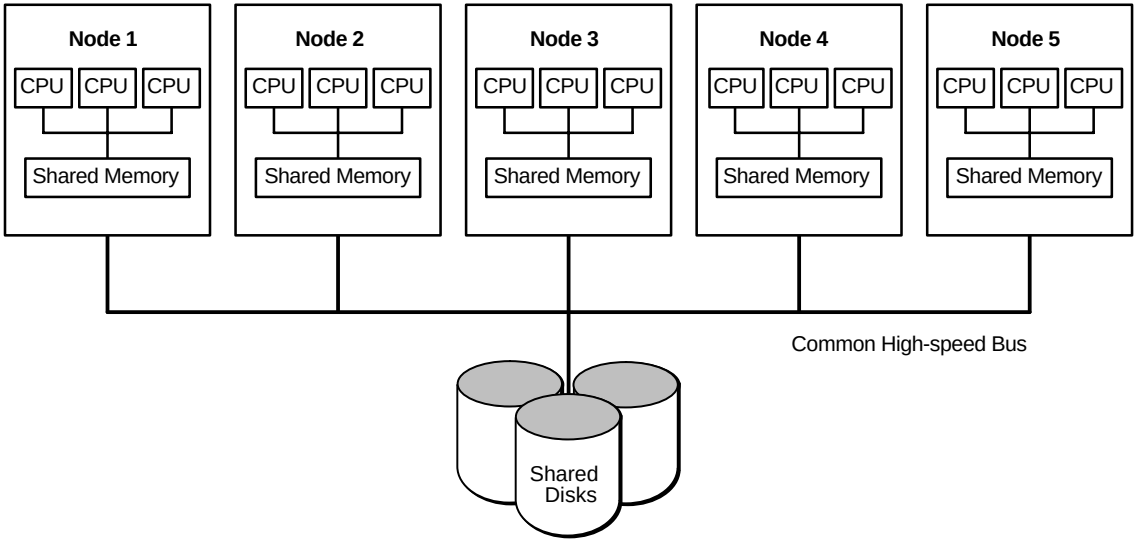
- Scalability is limited by bus bandwidth and latency, and by available memory

Shared Disk Systems

Shared disk systems are typically loosely coupled. Such systems, illustrated in Figure 3–2, have the following characteristics:

- Each node consists of one or more CPUs and associated memory
- Memory is not shared among nodes
- Communication occurs over a common high-speed bus
- Each node has access to the same disks and other resources
- A node can be an SMP if the hardware supports it
- Bandwidth of the high-speed bus limits the number of nodes (scalability) of the system

Figure 3–2 Loosely Coupled Shared Disk System



The cluster illustrated in Figure 3–2 is composed of multiple, tightly coupled nodes. The IDLM is required. Examples of loosely coupled systems are VAX clusters or Sun clusters.

Since memory is not shared among the nodes, each node has its own data cache. Cache consistency must be maintained across the nodes and a lock manager is

needed to maintain the consistency. Additionally, instance locks using the IDLM on the Oracle level must be maintained to ensure all nodes in the cluster see identical data.

There is additional overhead in maintaining the locks and ensuring data cache consistency. The effect on performance is dependent on the hardware and software components, such as the high-speed bus bandwidth through which the nodes communicate, and IDLM performance.

Parallel processing advantages of shared disk systems are:

- Shared disk systems permit high availability. All data is accessible even if one node dies.
- These systems have the concept of "one database", which is an advantage over shared nothing systems.
- Shared disk systems provide for incremental growth.

Parallel processing disadvantages of shared disk systems are:

- Inter-node synchronization is required, involving IDLM overhead and greater dependency on high-speed interconnect.
- If the workload is not partitioned well, there may be high synchronization overhead.
- There is operating system overhead of running shared disk software.

Note: Memory mapped hardware available in late 1998 will provide functionality to copy buffers directly from one user address on one node to another user address on another node.

Shared Nothing Systems

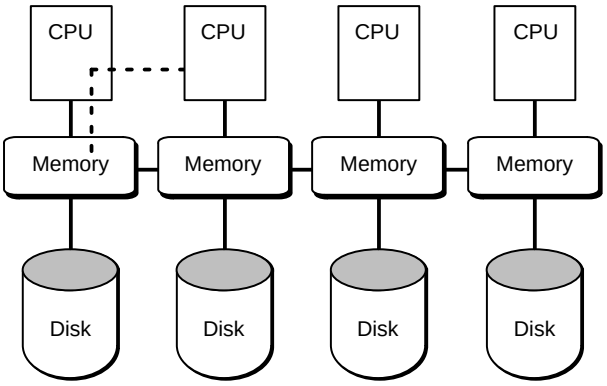
Shared nothing systems are typically loosely coupled. This section describes:

- Overview of Shared Nothing Systems
- Massively Parallel Systems
- Summary of Shared Nothing Systems

Overview of Shared Nothing Systems

In shared nothing systems, only one CPU is connected to a given disk. If a table or database is located on that disk, access depends entirely on the CPU that owns it. Figure 3–3 illustrates shared nothing systems:

Figure 3–3 Shared Nothing System



Shared nothing systems are concerned with access to disks, not access to memory. Nonetheless, adding more CPUs and disks can improve scaleup. OPS can access the disks on a shared nothing system as long as the operating system provides transparent disk access, but this access is expensive in terms of latency.

Massively Parallel Systems

Massively parallel (MPP) systems have these characteristics:

- MPP systems can range in size from only a few nodes to up to thousands of nodes
- The cost per processor may be extremely low because each node is an inexpensive processor
- Each node has associated non-shared memory
- Each node may have its own devices, but during failures other nodes can access the devices of the failed node
- Nodes are organized in a grid, mesh, or hypercube arrangement
- Oracle instances can potentially reside on any or all nodes

As mentioned, an MPP system can have as many as several thousand nodes. Each node may have its own Oracle instance with all the standard facilities of an instance. (An Oracle instance comprises the System Global Area and all the background processes.)

An MPP has access to a huge amount of real memory for all database operations (such as sorts or the buffer cache), since each node has its own associated memory. To avoid disk I/O, this advantage is important to long running queries and sorts. This is not possible for 32-bit machines which have 2GB addressing limits; total memory on MPP systems may be over 2GB. As with loosely coupled systems, cache consistency on MPPs must still be maintained across all nodes in the system. Thus, the overhead for cache management is still present. Examples of MPP systems are the nCUBE2 Scalar Supercomputer, the Unisys OPUS, Amdahl, Meiko, Pyramid, Smile, and the IBM SP.

Summary of Shared Nothing Systems

Shared nothing systems have advantages and disadvantages for parallel processing:

Advantages

- Shared nothing systems provide incremental growth
- System growth is practically unlimited
- MPPs are generally good for read-only, DSS applications
- Failure is local: if one node fails, the others stay up

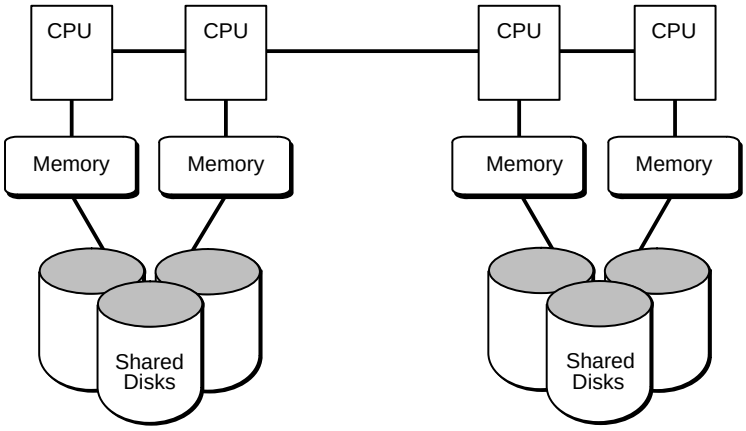
Disadvantages

- More coordination is required
- More overhead is required for processes working on a disk belonging to another node
- If there is a heavy workload of updates or inserts, as in online transaction processing systems, it may be worthwhile to consider data-dependent routing to reduce contention.

Shared Nothing /Shared Disk Combined Systems

A combined system can be very advantageous. This unites advantages of shared nothing and shared disk, while overcoming their respective limitations. Figure 3–4 illustrates a combined system:

Figure 3–4 Two Shared Disk Systems Forming a Shared Nothing System



Here, two shared disk systems are linked to form a system with the same hardware redundancies as a shared nothing system. If one CPU fails, the other CPUs can still access all disks.

Part II

Oracle Parallel Server Concepts

4

How Oracle Implements Parallel Processing

This chapter gives a high-level view of how the Oracle Parallel Server (OPS) provides high performance parallel processing. Key issues include:

- Enabling and Disabling Parallel Server
- Synchronization
- High Performance Features
- Cache Coherency

> **See Also:** Chapter 7, "Overview of Locking Mechanisms" for an understanding of the lock hierarchy within Oracle.

Enabling and Disabling Parallel Server

OPS can be enabled or disabled:

Oracle + Option	Parallel Server Disabled	Parallel Server Enabled	
		Single Node	Multiple Nodes
OPS not installed	Yes: default	No	No
OPS installed	Yes: default	Yes: Single Shared	Yes: Multiple Shared

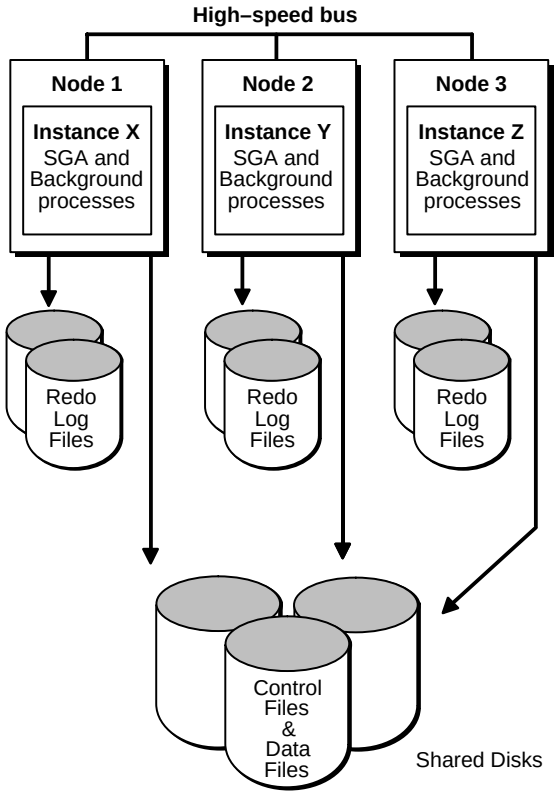
When parallel server is disabled, only one Oracle instance can mount or open the database. This mode is necessary to create and completely recover a database. It is useful to install OPS and disable it if standard Oracle functionality meets your needs. You can later enable OPS.

When OPS is enabled, one or more instances of a parallel server mount the same database. All instances mount the database and read from and write to the same

datafiles. *Single shared mode* describes an OPS configuration with only one instance. Global operations exist, but are not needed at the moment. The instance operates as though it is in a cluster with Integrated Distributed Lock Manager (IDLM) overhead, and so on, although there is no contention for resources. *Multiple shared mode* describes an OPS configuration with multiple instances running.

Figure 4–1 illustrates a typical OPS configuration with three instances on separate nodes accessing the same database.

Figure 4–1 Shared Mode Sharing Disks



Note: Each instance can access the redo log files of the other instances.

See Also: "Enabling Parallel Server and Starting Instances" on page 18-13.

Synchronization

Inter-node synchronization is an issue that does not need to be addressed in standard Oracle. But with OPS you must have a broad understanding of the dimensions in which synchronization must occur. Some of these include:

- Block Level Locking
- Row Level Locking
- Space Management
- System Change Number

In OPS exclusive mode, all synchronization is done within the instance. In shared mode, synchronization is accomplished with the help of the IDLM component.

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 needs to obtain an instance lock. Ownership of the lock is thus assigned to the instance.

Since OPS 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 IDLM ensures the validity of all copies of the block: these block-level locks are the buffer cache locks.

Block level locking occurs only when OPS is enabled. It is transparent to the user and to the application. (Row level locking also operates, whether OPS is enabled or disabled.)

See Also: Chapter 9, "Parallel Cache Management Instance Locks".

Row Level Locking

OPS, as well as single-instance Oracle, provides row level locking in addition to block level locking in the buffer cache. In fact, row level locks are stored within the block.

Consider the following example: Instance 1 reads file 2, block 10 to update row 1. Instance 2 also reads file 2, block 10, to update row 2. Here, instance 1 obtains an instance lock on block 10, then locks and updates row 1. (The row lock is implicit because of the UPDATE statement.)

Instance 2 then forces instance 1 to write the updated block to disk, and instance 1 relinquishes ownership of the lock on block 10 so instance 2 can assume ownership of it. Instance 2 then locks row 2 and performs an UPDATE.

Space Management

Free lists and free list groups optimize space management in OPS.

The problem of allocating space for inserts illustrates space management issues. When a table uses more space, how can you ensure no one else uses the same space? How can you ensure two nodes are not inserting into the same space on the same disk in the same file?

Consider the following example: Instance 1 reads file 2, block 10 to insert a row. Instance 2 reads file 3, block 20, to insert another row. Each instance proceeds to insert rows. If one block were responsible for assigning space for all these inserts, that block would constantly ping between the instances. Instance 1 would lose block ownership when instance 2 needs to insert, and so on. The situation involves a great deal of contention and performance suffers.

By contrast, free list groups make good space management possible. If two instances insert into the same object (such as a table), but each instance has its own set of free lists for that object, then contention for a single block is avoided. Each instance inserts into a different block belonging to the object.

System Change Number

In standard Oracle, the system change number (SCN) is maintained and incremented in the SGA by an exclusive mode instance. In OPS shared mode, the SCN must be maintained globally. Its implementation may vary from platform to platform. The SCN may be handled by the IDLM, by the Lamport SCN scheme, or by using a hardware clock or dedicated SCN server.

See Also: "Lamport SCN Generation" on page 4-7, "System Change Number (SC)" on page 10-4, and your Oracle system-specific documentation.

High Performance Features

OPS takes advantage of systems of linked processors sharing resources without sacrificing transaction processing features. The following sections discuss in more detail certain features that optimize performance on OPS.

- Fast Commits, Group Commits, and Deferred Writes
- Row Locking and Multiversion Read Consistency
- Online Backup and Archiving
- Sequence Number Generators
- Lamport SCN Generation
- Free Lists
- Free List Groups
- Disk Affinity

Within a single instance, Oracle uses a buffer cache in memory to reduce 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, OPS 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 more information about each of these high-performance features.

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)

Oracle may also perform unnecessary writes to disk caused by forced reads or forced writes.

See Also: "Detecting False Pinging" on page 15-14.

Row Locking and Multiversion Read Consistency

The Oracle row locking feature allows multiple transactions on separate nodes to lock and update different rows of the same data block, 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

OPS supports all Oracle backup features 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 can be overwritten. In OPS, 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. We strongly recommend operating production databases in ARCHIVELOG mode.

Cache Fusion

Cache Fusion improves inter-instance communication and reduces pinging for reader / writer cache coherency conflicts. When an instance requests a block for updating and another instance holds the block, Cache Fusion prepares a consistent read copy of the block and sends it directly to the requesting instance without pinging. Cache Fusion does this by copying blocks directly from the holding instance's memory cache to the requesting instance's memory cache.

See Also: For more information about the Cache Fusion architecture, please refer to "Cache Fusion Processing and the Block Server Process" on page 5-7. For information on locking mechanisms used in Cache Fusion, please refer to "The Role of Cache Fusion in Resolving Cache Coherency Conflicts" on page 20-2.

Sequence Number Generators

OPS allows users on multiple instances to generate unique sequence numbers with minimal cooperation and contention among instances.

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. IDLM locks coordinate sequences across instances in OPS.

Lamport SCN Generation

The System Change Number (SCN) is a logical timestamp 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.

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 7 seconds. You can examine the alert log after instance startup to see whether the Lamport SCN generation scheme is in use.

See Also: Your Oracle system-specific documentation.

Free Lists

Standard Oracle can use multiple free lists as a way to reduce contention on blocks. A free list is a list of data blocks, located in extents, that have free space. These blocks with free space are used when inserts or updates are made to a database object such as a table or a cluster. No contention among instances occurs when different instances' transactions insert data into the same table. This is achieved by locating free space for the new rows using *free space lists* that are associated with one or more instances. The free list may be from a common pool of blocks, or multiple free lists may be partitioned so specific extents in files are allocated to objects.

With a single free list when multiple inserts are taking place, single threading occurs as these processes attempt to allocate space from the free list. The advantage of using multiple free lists is that it allows processes to search a specific pool of blocks when space is needed. This reduces contention among users for free space.

> **See Also:** Chapter 11, "Space Management and Free List Groups".

Free List Groups

OPS can use free list groups to eliminate contention among instances for access to a single block containing free lists. By default, only one free list group is available. This means all free lists for an object reside in the segment header block.

Therefore, if multiple free lists reside in a single block in an OPS environment the block with the free lists could have pinging, or forced reads / writes among all the instances. To avoid this problem, free lists can be grouped, with one group assigned to each 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.

> **See Also:** Chapter 17, "Using Free List Groups to Partition Data" regarding proper use of free lists to achieve optimal performance in an OPS environment. Also read "Backing Up the Database" on page 21-12.

Disk Affinity

Disk affinity determines the instance that will perform parallelized DML or query operations. Affinity is especially important for parallel DML in OPS configurations. Affinity information that is consistent across statements improves buffer cache hit ratios and reduces forced reads / writes.

The granularity of parallelism for most PDML operations is by partition. For parallel query, however, granularity is by rowid. Parallel DML operations need a partition-to-instance mapping to implement affinity. The segment header of the partition is used to determine the affinity of the partition for MPP systems. You can achieve improved performance by having nodes access local devices. This provides a better buffer cache hit ratio for every node.

For other OPS configurations, a deterministic mapping of partitions to instances is used. Partition-to-instance affinity information is used to determine process allocation and work assignments for all OPS / MPP configurations.

> **See Also:** *Oracle8i Concepts* describes at length the concepts of Parallel Data Manipulation Language (PDML) and degree of parallelism. For a discussion of PDML tuning and optimizer hints, please see *Oracle8i Tuning*. Also refer to each installation and configuration guide for port-specific information on disk affinity.

Job and Instance Affinity

Use this feature to control which instances process which jobs. Using the package DBMS_JOB, you can distribute jobs across a cluster in a manner that makes the most sense given each job's functions. This improves load balancing and limits block contention since only the SNP processes of the selected instance can execute the job.

As an example, simultaneously using OPS and replication often results in pinging on the deferred transaction queue if all instances in a clustered environment propagate transactions from the deferred transaction queue. To limit activity against tables to only one instance, use DBMS_JOB to assign the work of processing jobs in the queue to a particular OPS instance.

Although the following examples use replication to illustrate job affinity, you can use this feature for other scenarios.

Using the DBMS_JOB Package

For this example, a constant in DBMS_JOB indicates "no mapping" among jobs and instances, that is, jobs can be executed by any instance.

DBMS_JOB.SUBMIT To submit a job to the job queue, use the following syntax:

```
DBMS_JOB.SUBMIT( JOB OUT BINARY_INTEGER,
WHAT IN VARCHAR2, NEXT_DATE IN DATE DEFAULTSYSDATE,
INTERVAL IN VARCHAR2 DEFAULT 'NULL',
```

```
NO_PARSE IN BOOLEAN DEFAULT FALSE,
INSTANCE IN BINARY_INTEGER DEFAULT ANY_INSTANCE,
FORCE IN BOOLEAN DEFAULT FALSE)
```

Use the parameters INSTANCE and FORCE to control job and instance affinity. The default value of INSTANCE is 0 (zero) to indicate that any instance can execute the job. To run the job on a certain instance, specify the INSTANCE value. Oracle displays error ORA-23319 if the INSTANCE value is a negative number or a NULL.

The FORCE parameter defaults to FALSE. If force is TRUE, any positive integer is acceptable as the job instance. If FORCE is FALSE, the specified instance must be running, or Oracle displays error number ORA-23428.

DBMS_JOB.INSTANCE To assign a particular instance to execute a job, use the following syntax:

```
DBMS_JOB.INSTANCE(  JOB IN BINARY_INTEGER,
INSTANCE IN BINARY_INTEGER,
FORCE IN BOOLEAN DEFAULT FALSE)
```

The FORCE parameter in this example defaults to FALSE. If the instance value is 0 (zero), job affinity is altered and any available instance can execute the job despite the value of force. If the INSTANCE value is positive and the FORCE parameter is FALSE, job affinity is altered only if the specified instance is running, or Oracle displays error ORA-23428.

If the FORCE parameter is TRUE, any positive integer is acceptable as the job instance and the job affinity is altered. Oracle displays error ORA-23319 if the INSTANCE value is negative or NULL.

DBMS_JOB.CHANGE To alter user-definable parameters associated with a job, use the following syntax:

```
DBMS_JOB.CHANGE(  JOB IN BINARY_INTEGER,
WHAT IN VARCHAR2 DEFAULT NULL,
NEXT_DATE IN DATE DEFAULT NULL,
INTERVAL IN VARCHAR2 DEFAULT NULL,
INSTANCE IN BINARY_INTEGER DEFAULT NULL,
 FORCE IN BOOLEAN DEFAULT FALSE )
```

Two parameters, INSTANCE and FORCE, appear in this example. The default value of INSTANCE is NULL indicating that job affinity will not change.

The default value of FORCE is FALSE. Oracle displays error ORA-23428 if the specified instance is not running and error ORA-23319 if the INSTANCE number is negative.

DBMS_JOB.RUN The FORCE parameter for DBMS_JOB.RUN defaults to FALSE. If force is TRUE, instance affinity is irrelevant for running jobs in the foreground process. If force is FALSE, the job can run in the foreground only in the specified instance. Oracle displays error ORA-23428 if force is FALSE and the connected instance is the incorrect instance.

```
DBMS_JOB.RUN( JOB IN BINARY_INTEGER,
FORCE IN BOOLEAN DEFAULT FALSE)
```

> **See Also:** For details about DBMS_JOB, please refer to the *Oracle8i Administrator's Guide* and the *Oracle8i Supplied Packages Reference.*

Transparent Application Failover

Application failover enables an application to automatically reconnect to a database if the connection is broken. Active transactions roll back, but the new database connection is identical to the original one. This is true regardless of how the connection was lost.

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.

> **See Also:** "Connection Load Balancing" on page 18-32 and "Recovery from Instance Failure" on page 22-2 and *Oracle8i Tuning.*

Cache Coherency

Cache coherency is the technique of keeping multiple copies of an object consistent. This section describes:

- Parallel Cache Management Issues
- Non-PCM Cache Management Issues

Parallel Cache Management Issues

With OPS, separate Oracle instances run simultaneously on one or more nodes using a technology called parallel cache management (PCM).

PCM uses IDLM locks (IDLM) to coordinate access to resources required by the instances of OPS. Rollback segments, dictionary entries, and data blocks are examples of database resources. The most often required database resources are data blocks.

Cache coherency is provided by the Parallel Cache Manager for the buffer caches of instances located on separate nodes. The set of global constant (GC_*) initialization parameters associated with PCM buffer cache locks are *not* the same locks as those used with the dictionary cache, library cache, and so on.

PCM ensures that a master copy data block in an SGA has identical copies in other SGAs requiring a copy of the master. Thus, the most recent copy of the block in all SGAs 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 all existing copies in other buffer caches are no longer current. New copies can be obtained after the modification operation completes.

PCM enforces cache coherency while minimizing I/O and use of the IDLM. I/O and lock operations for cache coherency are only done 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 OPS instance can share access to a set of data blocks for reading purposes without additional instance 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.

In shared mode, the IDLM maintains instance lock status. In exclusive mode, all locks are local and the IDLM does not coordinate database resources.

Instances use instance locks to indicate ownership of a resource master copy. When an instance becomes a database resource master or "owner", it also inherently becomes owner of the instance 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 *gives up* the instance 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 instance lock.

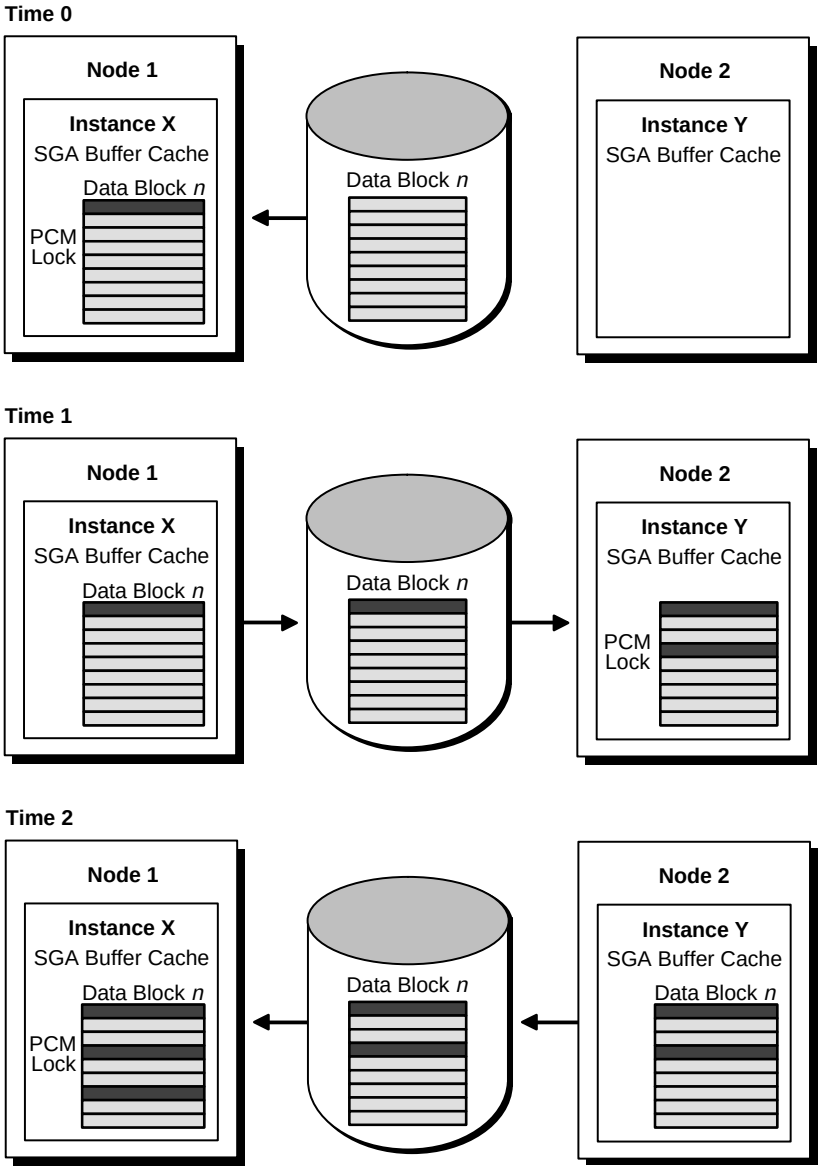
Note: Transactions and parallel cache management are autonomous mechanisms in Oracle. PCM locks function independently of any form of transaction lock.

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

- Instance X becomes the owner of a PCM lock covering data block *n* containing row 1 and updates the row
- Instance Y requests the block to update row 4
- Instance X writes the data block to disk and releases the PCM lock
- Instance Y becomes the owner of the block and the PCM lock and then updates row 4
- Instance X requests the block to update row 7
- Instance Y writes the data block to disk and releases the block and the PCM lock
- Instance X becomes the owner of the block and PCM lock and updates row 7
- Instance X commits its transaction and still owns the PCM lock and the master copy of the block until another instance requests the block

See Also: "How Buffer State and Lock Mode Change" on page 9-10.

Figure 4–2 Multiple Instances Updating the Same Data Block



PCM Lock and Row Lock Independence

PCM locks and row locks operate independently. An instance can disown a PCM lock without affecting row locks held in the set of blocks covered by the PCM lock. A row lock is acquired during a transaction. A database resource such as a data block acquires a PCM lock when it is read for update by an instance. During a transaction, a PCM 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 PCM locks, and PCM lock operations do not depend on individual transactions committing or rolling back.

Instance Lock Modes

An instance can acquire the instance 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 instance lock covering the block, the first instance uses the IDLM to request that the second instance disown the instance lock, writing the block(s) to disk if necessary.

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

  Multiple instances can own an instance lock in shared mode as long as they only intend to read, not modify, blocks covered by that instance 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 instance lock operations to request the block from another instance. This means instances do not have to disown instance 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.

See Also: Chapter 15, "Allocating PCM Instance Locks", for a detailed description of allocating PCM locks for datafiles.

Non-PCM Cache Management Issues

OPS ensures that all standard Oracle caches are synchronized across instances. Changing a block on one node, and its ramifications for the other nodes, is a familiar example of synchronization. However, synchronization has broader implications.

Understanding how OPS 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. OPS handles this automatically through the IDLM. 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 command forces other caches to be flushed, but the brief effect this has on performance does not necessarily diminish the advantage of having multiple caches.

See Also: "Space Management" on page 4-4, and "System Change Number" on page 4-4 for additional examples of non-PCM cache management issues.

5

Oracle Instance Architecture for Oracle Parallel Server

*[Architecture] is music in space, as if it were a frozen mu*sic...

— Schelling, *Philosophie der Kunst*

This chapter explains features of the Oracle Parallel Server (OPS) architecture that differ from an Oracle server in exclusive mode.

- Overview
- Characteristics of OPS Multi-instance Architecture
- System Global Area
- Background Processes
- Configuration Guidelines for Oracle Parallel Server

Overview

Each Oracle instance in an OPS architecture has its own:

- System global area (SGA)
- Background processes
- ORACLE_SID (can be the same for each instance).
- Set of redo logs

All instances in an OPS environment share or need to access the same sets of:

- Data files

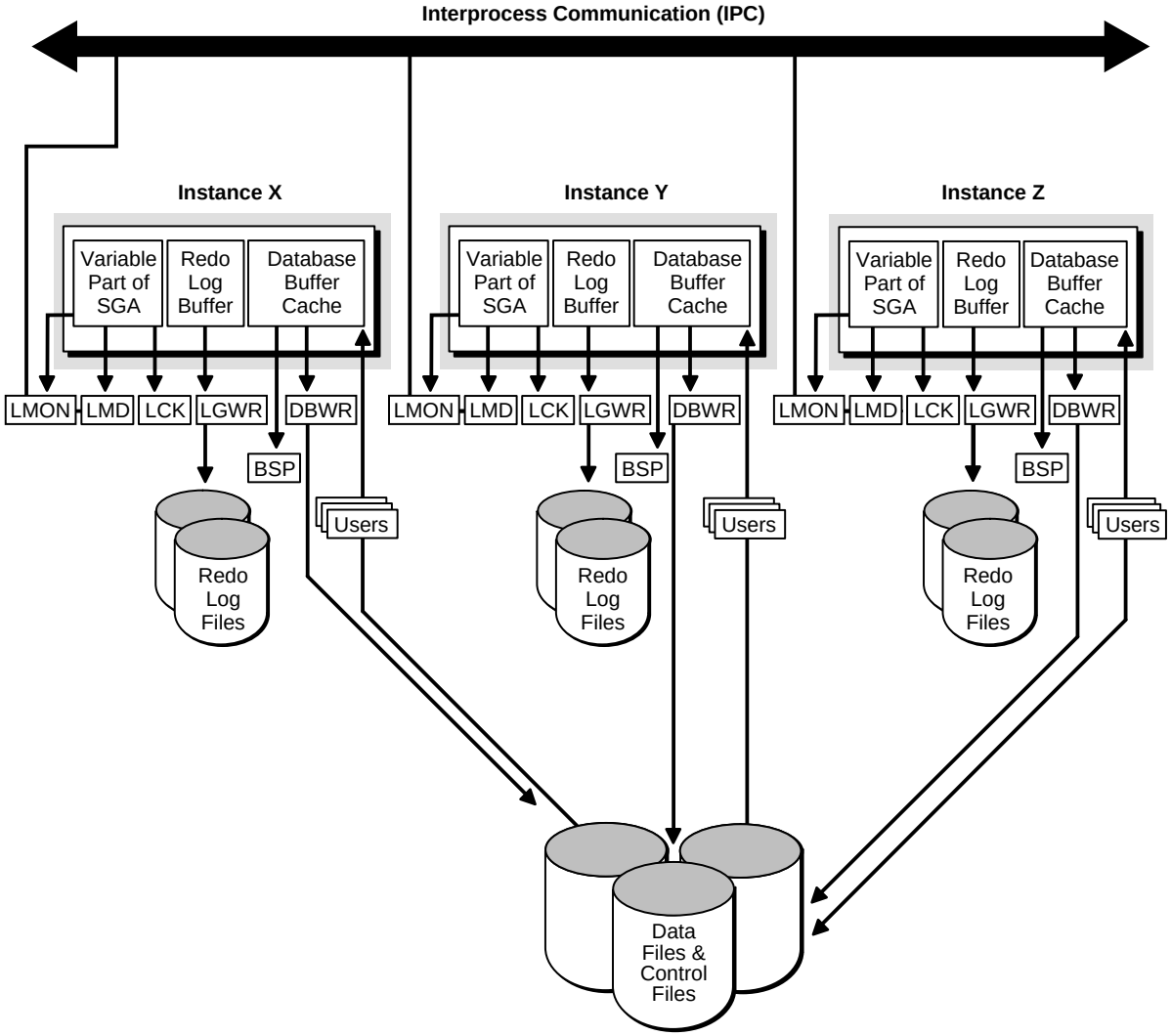
- Control files
- Redo logs

The OPS instance contains:

- An additional PCM lock area in its SGA to coordinate shared resource or "lock element" use
- The Integrated Distributed Lock Manager (IDLM) component, an area for global locks and resources
- Additional background processes LCK*n* to coordinate shared resource locking among multiple instances in a parallel server
- Additional background processes LMON and LMD0 to manage global locks and resources

The basic OPS components appear in Figure 5–1. DBWR processes are shown writing data, users are reading data. The background processes LMD and LCK, as well as foreground (FG) processes, communicate directly from one instance to another by way of the interconnect.

Figure 5–1 Basic Elements of Oracle Parallel Server



See Also: "Memory Structures and Processes" in *Oracle8i Concepts.*

Characteristics of OPS Multi-instance Architecture

The characteristics of OPS can be summarized as:

- An Oracle instance can be started on one or more nodes in the network
- Each instance has a separate System Global Area (SGA) and set of background processes
- All instances share the same datafiles and control file
- Each instance has its own set of redo log files

> **Note:** Redo logs must be accessible to all instances in case of instance failure. On some MPP platforms, a redo server exists so only one set of redo logs is necessary for the whole OPS system

- Archived logs are private, but must be accessible to all instances for media recovery
- All instances can execute transactions concurrently against the same database, and each instance can have multiple users executing transactions
- Row level locking is preserved

A parallel server is administered in the same manner as a non-parallel server, except that you must connect to a particular instance to perform certain administrative tasks. For example, creating users or objects can be done from any single instance.

Applications accessing the database can run on the same nodes as instances of a parallel server or on separate nodes, using the client-server architecture. A parallel server can be part of a distributed database system. Distributed transactions access data in a remote database in the same manner, regardless of whether the datafiles are owned by a standard Oracle Server in exclusive mode or by a parallel server in exclusive or shared mode.

Other non-Oracle processes can run on each node, or you can dedicate the entire system or part of the system to Oracle. For example, a parallel server and its applications might occupy three nodes of a five-node configuration, while the other two nodes are used for non-Oracle applications.

System Global Area

Each instance of a parallel server has its own System Global Area (SGA). The SGA has the following memory structures:

- Buffer cache for data blocks
- Dictionary cache for data dictionary information
- Redo log buffer for redo entries
- Shared pool containing the shared SQL and shared PL/SQL areas
- Instance lock area (only in a parallel server)

Data sharing among SGAs in OPS is controlled by parallel cache management using parallel cache management (PCM) locks.

Copies of the same data block can be present in several SGAs at the same time. PCM locks ensure that the database buffer cache is kept consistent for all instances. It thus ensures readability by one instance of changes made by other instances.

Each instance has a shared pool that can only be used by the user applications connected to that instance. If the same SQL statement is submitted by different applications using the same instance, it is parsed and stored once in that instance's SGA. If that same SQL statement is also submitted by an application on another instance, then the other instance also parses and stores the statement.

> **See Also:** Chapter 9, "Parallel Cache Management Instance Locks".

Background Processes

Each instance in OPS has its own set of background processes that are identical to the background processes of a single server in exclusive mode. The DBWR, LGWR, PMON, and SMON processes are present for every instance; the optional processes, ARCH, CKPT, D*nnn* and RECO, can be enabled by setting initialization parameters. In addition to the standard background processes, each instance of OPS has at least one lock process, LCK0. You can enable additional lock processes if needed.

In OPS, IDLM also uses the LMON and LMD0 processes. LMON manages instance and failures and associated recovery for the IDLM. In particular, LMON handles the part of recovery associated with global locks. LMD processes handle remote lock requests such as those originating from other instances. LMD also handles deadlock detection. The LCK process manages locks used by an instance and coordinates requests from other instances for those locks.

When an instance fails in shared mode, another instance's SMON detects the failure and recovers for the failed instance. The LCK process of the instance doing the recovery cleans up outstanding PCM locks for the failed instance.

> **See Also:** "The LCK Process" on page 7-6 and "GC_* Initialization Parameters" on page 9-13.

Foreground Lock Acquisition

Foreground processes communicate lock requests directly to remote LMD processes. Foreground processes send request information such as the resource name it is requesting a lock for and the mode in which it is needs the lock.

The IDLM processes the request asynchronously, so the foreground process waits for the request to complete before closing the request.

> **See Also:** For more information about how these requests are processed, please refer to "Asynchronous Traps (ASTs) Communicate Lock Request Status" on page 8-2.

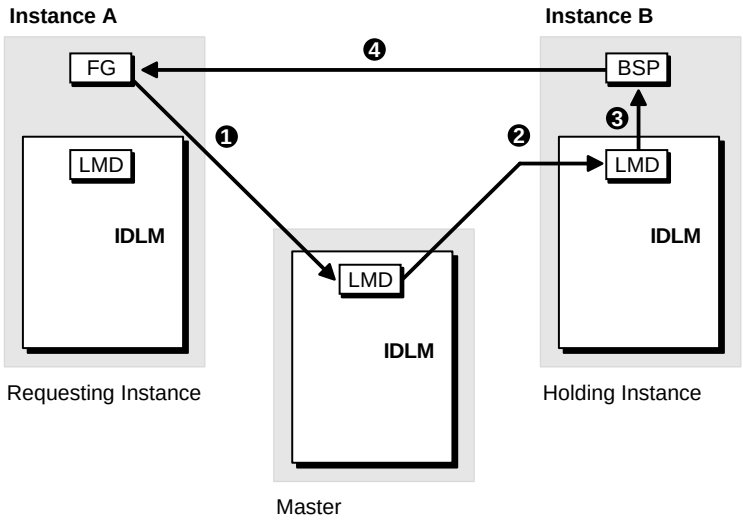
Cache Fusion Processing and the Block Server Process

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.

Cache Fusion uses the Block Server Process (BSP) 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 5–2 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 OPS candidates, such as OLTP and hybrid applications.

Figure 5–2 Consistent Read Server Processing



1. 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.

2. 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.
3. 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.
4. BSP prepares and sends the requested block to the requestor's FG process.

Configuration Guidelines for Oracle Parallel Server

When setting up OPS, observe the guidelines in Table 5–1:

Table 5–1 Parallel Server Configuration Guidelines

Configuration Issue	Guidelines
Version	Ensure that the same Oracle version exists on all the nodes.
Links	UNIX soft or hard inks ("aliases") to executables are not recommended for OPS. If the single node containing the executables fails, none of the nodes can operate.
Initialization parameters	Keep initialization parameters in a single file. These parameters should be identical across all OPS instances. Include this file in the individual initialization files of the different instances using the IFILE option. You should keep instance specific parameters such as ROLLBACK SEGMENTS, THREAD INSTANCE, and so on, in the local instance parameter file that also contains the IFILE. The IFILE points to the larger common file that contains all other parameters that should remain identical.
Control files	Must be accessible from all instances.
Data files	Must be accessible from all instances.
Log files	Must be located on the same set of disks as control files and data files. Although the redo log files are independent for each instance, each log file must still be accessible by all instances to allow recovery.
NFS	You can use NFS to enable access to Oracle executables, but not access to database files or log files. If you are using NFS, the serving node is a single point of failure.
Archived redo log files	Must be accessible from all instances.

6

Oracle Database Architecture for the Parallel Server

This chapter describes features of the Oracle database architecture pertaining to multiple instances of OPS.

- File Structures
- The Data Dictionary
- The Sequence Generator
- Rollback Segments

File Structures

The following sections describe the features of control files, datafiles, and redo log files that apply to OPS.

- Control Files
- Datafiles
- Redo Log Files

Control Files

All OPS instances access the same control files. The control files hold values of *global constant* initialization parameters, such as GC_FILES_TO_LOCKS, some of which must be identical for all instances running concurrently. As each instance starts, Oracle compares the global constant initialization values in a common parameter file (or in parameter files for each instance) with those in the control file, and generates a message if the values are different.

Note: There is only one control file per cluster. However, the control file might be mirrored across other instances so each instance accesses a copy of the same control file.

See Also: For more information on parameters, please see "Parameters that Must Be Identical on All Instances" on page 18-11, and "Initialization Parameters" in Appendix A. Also see *Oracle8i Concepts*.

Datafiles

All OPS instances access the same datafiles. Database files are the same for Oracle in parallel mode as in exclusive mode. You do not have to change the datafiles to start Oracle in exclusive or parallel mode.

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

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

If verification fails for any instance, 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 datafiles. It also has a LOCAL option that makes the current instance verify access.

ALTER SYSTEM CHECK DATAFILES makes the online datafiles 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 datafiles.

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

> **See Also:** For more information, please see Chapter 17, "Using Free List Groups to Partition Data", "Access to Datafiles for Instance Recovery" on page 22-4, and "Setting GC_FILES_TO_LOCKS: PCM Locks for Each Datafile" on page 15-6. For more information about relative file numbers, see *Oracle8i Concepts*.

Redo Log Files

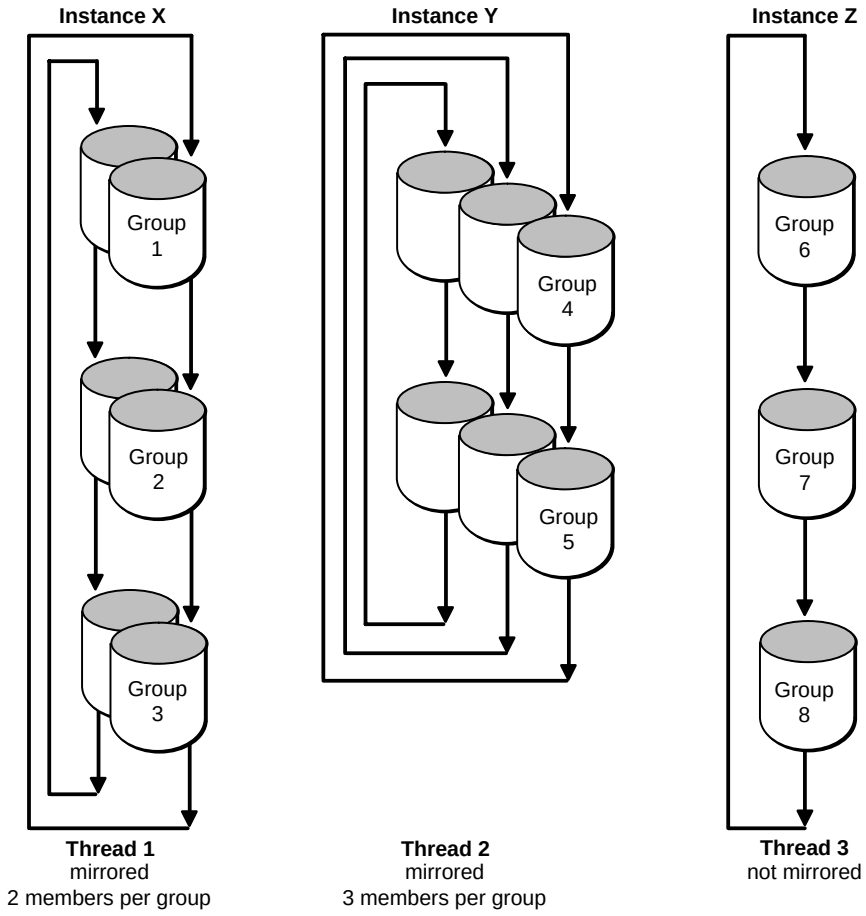
In OPS, each instance writes to its own set of online redo log files. The redo written by a single instance is called a *thread* of redo. Each online redo log file is associated with a particular thread number. When an online redo log is archived, Oracle records its thread number to identify it during 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 6–1 shows the threads of redo for three instances of OPS.

Figure 6–1 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.

Group numbers must be unique within the database, therefore they are not unique within a thread. However, the order of assigning groups to threads, and threads to instances is arbitrary.

For example, although in Figure 6–1 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 OPS 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 OPS, set the value of MAXLOGHISTORY higher than you normally would in single instance Oracle. This is because in OPS, 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: For more information, see "Recovery Structures" in *Oracle8i Concepts* for a full description of multiplexed redo log files, "Archiving the Redo Log Files" and "Checkpoints and Log Switches" on page 21-2, and "Recovery from Media Failure" on page 22-6.

The Data Dictionary

Each instance of OPS has a dictionary cache, or row cache, containing data dictionary information in its SGA. The data dictionary structure is the same for Oracle instances in OPS as for instances in exclusive mode. Instance locks coordinate the data dictionary activity of multiple instances.

The Sequence Generator

This section describes the CREATE SEQUENCE statement and its options.

- The CREATE SEQUENCE Statement
- The CACHE Option
- The ORDER Option

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.

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

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:** For more information about the CREATE SEQUENCE and CYCLE options, please refer to the *Oracle8i SQL Reference*.

The CACHE Option

The CACHE option of CREATE SEQUENCE pre-allocates sequence numbers and retains them in an instance's SGA 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 OPS environment.

Note: OPS 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.

Rollback Segments

This section describes rollback segments as they relate to OPS.

- Rollback Segments in OPS
- Parameters Controlling Rollback Segments
- Public and Private Rollback Segments
- How Instances Acquire Rollback Segments

Rollback Segments in OPS

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 OPS 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 shutdown 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 in exclusive mode.

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.

OPS 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:** "Creating Additional Rollback Segments" on page 14-5 for information about the rollback segments that are required when you create a database and the *Oracle8i Administrator's Guide* for information about contention for a rollback segment and 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 additional 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: Please see "Monitoring Rollback Segments" on page 14-6 and the discussion on "Data Blocks, Extents, and Segments" in *Oracle8i Concepts* for more information.

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 shutdown 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_RBS 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 initialization parameters to determine how many rollback segments to acquire as shown in the following equation:

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

The *total_rollback_segments_required* number is rounded up.

At startup, an instance attempts to acquire rollback segments by:

- First acquiring 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. I*n 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.

See Also: For more information, please see "Monitoring Rollback Segments" on page 14-6 and the *Oracle8i SQL Reference.*

7

Overview of Locking Mechanisms

This chapter provides an overview of Oracle Parallel Server's (OPS) internal locking mechanisms by covering the following topics:

- Differentiating Oracle Locking Mechanisms
- Cost of Locks
- Oracle Lock Names
- Coordination of Locking Mechanisms by the IDLM

Differentiating Oracle Locking Mechanisms

This section covers the following topics:

- Overview
- Local Locks
- Instance Locks
- The LCK Process
- The LMON and LMD0 Processes

Overview

You must understand locking mechanisms to effectively harness parallel processing and parallel database capabilities. You can influence each type of locking by setting initialization parameters, administering the system, and by designing efficient applications. Not using locks effectively causes your system to spend so much time synchronizing shared resources that you do not achieve speedup and scaleup. Your parallel system could even suffer performance degradation.

OPS uses locks for two primary purposes:

- Transaction isolation
- Cache coherency

Transaction locks implement row level locking for transaction consistency. Row level locking is supported in both single instance Oracle and OPS.

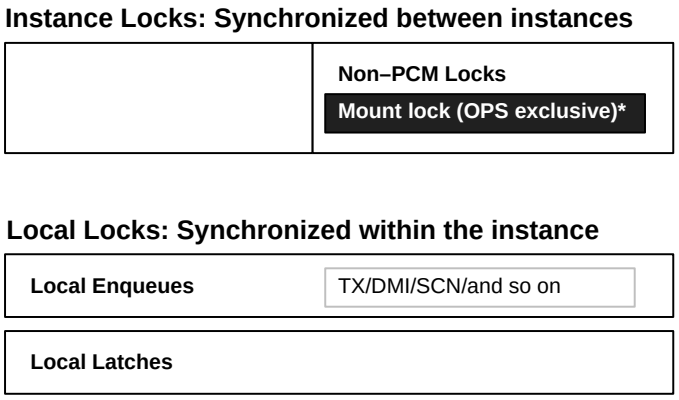
Instance locks (also commonly known as *distributed locks*) guarantee cache coherency. They ensure the consistency of data and other resources distributed among multiple instances belonging to the same database. Instance locks include PCM and non-PCM locks.

> **See Also:** For more information about Oracle locks, please refer to Chapter 8, "Integrated Distributed Lock Manager" and to *Oracle8i Concepts*.

Local Locks

Figure 7–1 shows latches and enqueues: locking mechanisms that are synchronized within a single instance. These are used in single instance Oracle and in OPS whether parallel server is enabled or disabled.

Figure 7–1 Locking Mechanisms: Oracle and OPS Disabled



* The mount lock is obtained if the Parallel Server Option has been linked in to your Oracle executable.

Latches

Latches are simple, low level serialization mechanisms that protect in-memory data structures in the SGA. Latches do not protect datafiles. They are entirely automatic and are held for a very short time in exclusive mode. Being local to the node, internal locks and latches do not provide internode synchronization.

Enqueues

Enqueues are shared memory structures that serialize access to database resources. These locks can be local to one instance or global to a database. They are associated with a session or transaction and can be in any mode:

- Shared
- Exclusive
- Protected read
- Protected write
- Concurrent read
- Concurrent write
- Null

Enqueues are held longer than latches, have more granularity and more modes, and protect more database resources. For example, if you request a table lock, or a DML lock, you receive an "enqueue".

Certain enqueues are local to single instances when OPS is disabled. But with OPS enabled, enqueues must be maintained on a system-wide level. Enqueues are managed by the Integrated Distributed Lock Manager (IDLM).

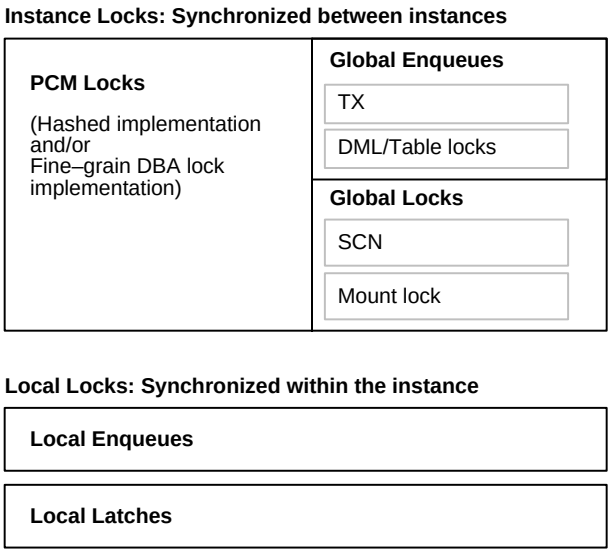
When OPS is enabled, most local enqueues become global enqueues. This is reflected in Figure 7–1 and Figure 7–2. They appear as enqueues in the fixed tables—no distinction is made between local and global enqueues. Global enqueues are handled in a distributed fashion.

Note: Transaction locks are simply a subset of enqueues.

Instance Locks

Figure 7–2 illustrates the instance locks used by OPS. In OPS implementations, the status of all Oracle locking mechanisms is tracked and coordinated by the IDLM.

Figure 7–2 Locking Mechanisms: Parallel Server Enabled



Instance locks (other than the mount lock) only come into existence if you start an Oracle instance with OPS enabled. They synchronize between instances, communicating the current status of a resource among the instances of OPS.

Instance locks are held by background processes of instances rather than by transactions. An instance *owns* an instance lock that *protects* a resource, such as a data block or data dictionary entry, when the resource enters its SGA.

To ensure cache coherency, the IDLM handles locking only for resources accessed by more than one instance of OPS. The IDLM communicates requests for instance locks and the status of the locks between lock processes of each instance. There are several views associated with the IDLM as described in Table 7–1.

Table 7–1 IDLM Views

IDLM View Name	Content
V$DLM_LOCKS	Lists information on blocking and blocked locks currently known to the IDLM.
V$DLM_ALL_LOCKS	Shows all locks and their states.
V$DLM_RESS	Shows all resources and their states.

There are two type of Instance locks: Parallel Cache Management (PCM) and Non-PCM locks.

PCM Locks

PCM locks are instance locks covering one or more data blocks (table or index blocks) in the buffer cache. PCM locks do not lock rows for transactions and are implemented in two ways:

Fine grain locking	This is the default implementation where PCM locks are assigned to blocks on a dynamic basis.
Hashed locking	This is where PCM locks are statically assigned to blocks in the datafiles.

With hashed locking, an instance never *disowns* a PCM lock unless another instance asks for it. This minimizes the overhead of instance lock operations in systems with relatively low contention for resources. With fine grain locking, once the block is released, the lock is released. Non-PCM locks *are* disowned.

Non-PCM Locks

There are many different types of non-PCM locks. These control access to data and control files, control library and dictionary caches, and perform various types of communication between instances. These locks do not protect datafile blocks. Examples are DML enqueues (table locks), transaction enqueues, and DDL or dictionary locks. The System Change Number (SCN), and the mount lock are global locks, not enqueues.

Note: The context of OPS 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 instance locks, such as SCN locks, mount locks, and PCM locks.

Many More PCM Locks Than Non-PCM Locks

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

You can control PCM locks in detail by setting initialization parameters to allocate the number desired. However, you have almost no control over non-PCM 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-PCM locks will still persist.

See Also: For more information, please see Chapter 16, "Ensuring IDLM Capacity for Resources and Locks".

The LCK Process

In OPS, the LCK process provides inter-instance locking. LCK manages most locks used by an instance and coordinates requests for those locks by other instances. LCK maintains all PCM locks, hashed or fine grain, and some of the non-PCM locks, such as row cache or library cache locks. LCK handles PCM as well as non-PCM locks.

Although instance locks are mainly handled by LCK, some instance locks are directly acquired by other background or shadow foreground processes. In general, if a background process such as LCK owns an instance lock, it is for the entire instance. If a foreground process owns an instance lock, it is just for that particular process. For example, the log writer (LGWR) obtains the SCN instance lock, the database writer (DBWR) obtains the media recovery lock. The bulk of these locks, however, are handled by LCK.

Foreground processes obtain transaction locks, LCK does not. Transaction locks are associated with the session / transaction unit, not with the process.

See Also: For more information about LCK, please refer to *Oracle8i Concepts.*

The LMON and LMD0 Processes

The LMON and LMD0 processes implement the global lock management subsystem of OPS. LMON performs lock cleanup and lock invalidation after the death of an Oracle shadow process or another Oracle instance. It also reconfigures and redistributes the global locks as OPS instances are started and stopped.

The LMD0 process handles remote lock requests for global locks, that is, lock requests originating from another instance for a lock owned by the current instance. All global lock messages directed to an OPS instance are handled by the LMD0 process of that instance.

Cost of Locks

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

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

In general, instance locks and global enqueues have an equivalent effect on performance. When OPS is disabled, all enqueues are local. When OPS is enabled, most enqueues are global.

Table 7–2 dramatizes the *relative expense* of latches, enqueues, and instance locks. The elapsed time required per lock varies by system. Values used in the "Actual Time Required" column of this table are only examples.

Table 7–2 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)
Instance Locks (or Global Enqueues)	1/10 second	100,000 minutes (69 days)

Microseconds, milliseconds, and tenths of a second may seem like negligible units of time. However, *imagine* the cost of locks using *grossly exaggerated values* such as those listed in the "Relative Time Required" column. This should make the need to carefully calibrate lock use in your systems and applications more obvious. In a large OLTP implementation, for example, you should avoid unregulated instance lock use. Imagine waiting hours or days to complete a transaction!

Stored procedures are available for analyzing the number of PCM locks an application uses if it performs particular functions. You can set values for your initialization parameters and then call the stored procedure to see the projected expenditure in terms of locks.

> **See Also:** For more information, please refer to Chapter 15, "Allocating PCM Instance Locks" and Chapter 16, "Ensuring IDLM Capacity for Resources and Locks".

Oracle Lock Names

This section covers the following topics:

- Lock Name Format
- PCM Lock Names
- Non-PCM Lock Names

Lock Name Format

All Oracle enqueues and instance locks are named using one of the following formats:

type ID1 ID2

or *type*, *ID1*, *ID2*

or *type* (*ID1*, *ID2*)

Where:

type	A two-character type name for the lock type, as described in the V$LOCK table, and listed in Table 7–3 and Table 7–4.
ID1	The first lock identifier, used by the IDLM. The convention for this identifier differs from one lock type to another.
ID2	The second lock identifier, used by the IDLM. The convention for this identifier differs from one lock type to another.

For example, a space management lock might be named ST 1 0. A PCM lock might be named BL 1 900.

The V$LOCK table lists local and global Oracle enqueues currently held or requested by the local instance. The "lock name" is actually the name of the resource; locks are taken out against the resource.

PCM Lock Names

All PCM locks are Buffer Cache Management locks.

Table 7–3 PCM Lock Type and Name

Type	Lock Name
BL	Buffer Cache Management

The syntax of PCM lock names is *type ID1 ID2*, where:

type	Is always BL because PCM locks are buffer locks.
ID1	Is the block class.
ID2	For fixed locks, *ID2* is the lock element (LE) index number obtained by hashing the block address (see the V$LOCK_ELEMENT fixed view). For releasable locks, *ID2* is the database address of the block.

Sample PCM lock names are:

BL (1, 100)	This is a data block with lock element 100.
BL (4, 1000)	This is a segment header block with lock element 1000.

BL (27, 1)	This is an undo segment header with rollback segment #10. The formula for the rollback segment is 7 + (10 * 2).

Non-PCM Lock Names

Non-PCM locks have many different names.

Table 7–4 Non-PCM Lock Types and Names

Type	Lock Name
CF	Controlfile Transaction.
CI	Cross-Instance Call Invocation.
DF	Datafile.
DL	Direct Loader Index Creation.
DM	Database Mount.
DX	Distributed Recovery.
FS	File Set.
KK	Redo Log "Kick".
IN	Instance Number.
IR	Instance Recovery.
IS	Instance State.
MM	Mount Definition.
MR	Media Recovery.
IV	Library Cache Invalidation.
L[A-P]	Library Cache Lock.
N[A-Z]	Library Cache Pin.
Q[A-Z]	Row Cache.
PF	Password File.
PR	Process Startup.
PS	Parallel Slave Synchronization.
RT	Redo Thread.
SC	System Commit Number.

Table 7–4 Non-PCM Lock Types and Names

Type	Lock Name
SM	SMON.
SN	Sequence Number.
SQ	Sequence Number Enqueue.
SV	Sequence Number Value.
ST	Space Management Transaction.
TA	Transaction Recovery.
TM	DML Enqueue.
TS	Temporary Segment (also Table-Space).
TT	Temporary Table.
TX	Transaction.
UL	User-Defined Locks.
UN	User Name.
WL	Begin written Redo Log.
XA	Instance Registration Attribute Lock.
XI	Instance Registration Lock.

See Also: Please refer to the *Oracle8i Reference* for descriptions of non-PCM locks.

Coordination of Locking Mechanisms by the IDLM

The IDLM component is a *distributed resource manager* that is *internal* to OPS. This section explains how the IDLM coordinates locking mechanisms that are *internal* to Oracle. Chapter 8, "Integrated Distributed Lock Manager" presents a detailed description of IDLM features and functions.

This section covers the following topics:

- The IDLM Tracks Lock Modes
- The Instance Maps Database Resources to IDLM Resources
- How IDLM Locks and Instance Locks Relate
- The IDLM Provides One Lock Per Instance on a Resource

The IDLM Tracks Lock Modes

In OPS implementations, the IDLM facility maintains an inventory of Oracle instance locks and global enqueues held against system resources. The IDLM acts as a referee when conflicting lock requests arise.

In Figure 7–3 the IDLM is represented as an inventory sheet listing resources and the current status of locks on each resource across OPS. Locks are represented as follows: S for shared mode, N for null mode, X for exclusive mode.

Figure 7–3 The IDLM Inventory of Oracle Resources and Locks

Integrated DLM

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

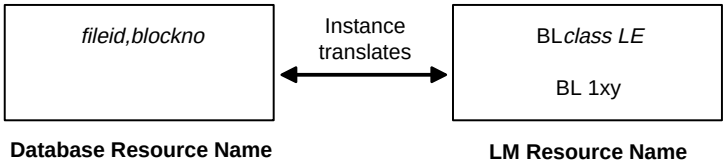
This inventory includes all instances. For example, resource BL 1, 101 is held by three instances with shared locks and three instances with null locks. Since the table reflects up to 6 locks on one resource, at least 6 instances are evidently running on this system.

The Instance Maps Database Resources to IDLM Resources

Oracle database resources are mapped to IDLM resources, with the necessary mapping performed by the instance. For example, a hashed lock on an Oracle database block with a given data block address (such as file 2 block 10) becomes 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 (DBA) is translated from the Oracle resource level to the IDLM resource level; the hashing function used is dependent on GC_* parameter settings. The IDLM resource name identifies the physical resource in views such as V$LOCK.

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

Figure 7–4 Database Resource Names Corresponding to IDLM Resource Names

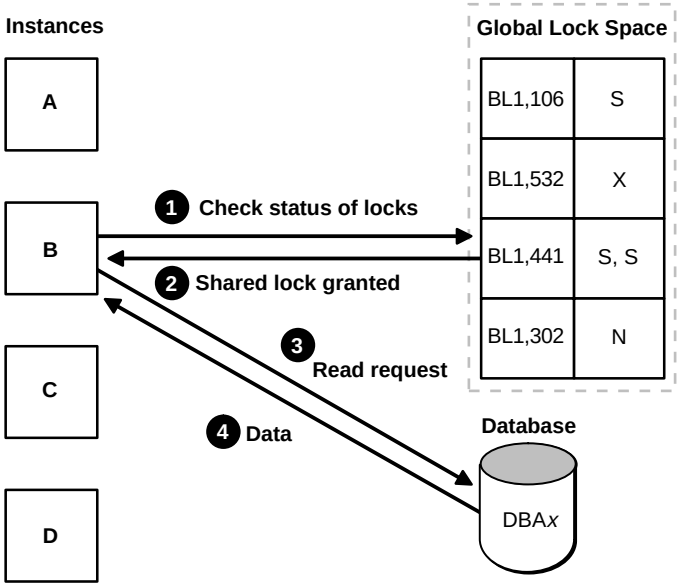


How IDLM Locks and Instance Locks Relate

Figure 7–5 illustrates how IDLM locks and PCM 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 data. The instance translates the block's database resource name to the IDLM resource name, and asks the IDLM for a shared lock in order to read the data.

As illustrated in the following conceptual diagram, the IDLM checks outstanding locks on the granted queue and determines there are already two shared locks on resource BL1,441. Since shared locks are compatible with read-only requests, the IDLM 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 7–5 The IDLM Checks Status of Locks



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

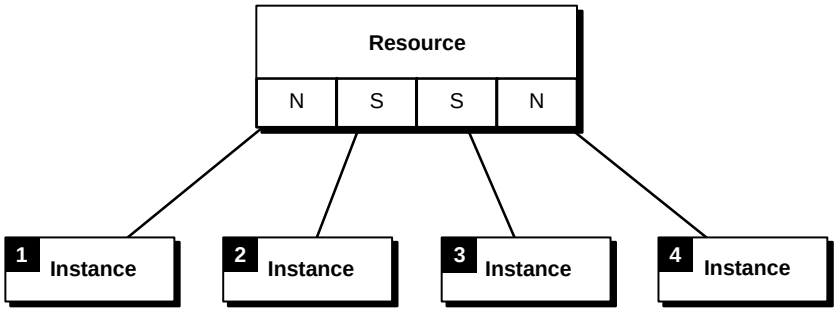
If the required block already had an exclusive lock on it from another instance, then instance B would have to wait for this to be released. The IDLM would place the shared lock request from instance B on the convert queue. The IDLM would then notify the instance when the exclusive lock was removed and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

The term *IDLM lock* refers simply to the IDLM's notations for tracking and coordinating the outstanding locks on a resource.

The IDLM Provides One Lock Per Instance on a Resource

The IDLM provides one lock per instance on a PCM resource. As illustrated in Figure 7–6, if you have a four-instance system and require a buffer lock on a single resource, you actually have four locks—one per instance.

Figure 7–6 Resources Have One Lock Per Instance



The number of non-PCM locks may depend on the type of lock.

> **See Also:** For more information, please see Chapter 10, "Non-PCM Instance Locks".

8

Integrated Distributed Lock Manager

This chapter explains the role of the Integrated Distributed Lock Manager (IDLM) in controlling access to Oracle Parallel Server (OPS) resources by covering the following topics:

- What Is the Integrated Distributed Lock Manager?
- The IDLM Grants and Coordinates Resource Lock Requests
- IDLM Lock Modes: Resource Access Rights
- IDLM Features

What Is the Integrated Distributed Lock Manager?

The IDLM component of Oracle8 maintains a list of system resources and provides locking mechanisms to control allocation and modification of Oracle resources. IDLM resources are logical concepts; structures of data. The IDLM does *not* control access to tables or objects in the database itself. Every process interested in a database resource protected by the IDLM must open a lock on the resource.

OPS uses the IDLM to coordinate concurrent access across multiple instances to resources such as data blocks and rollback segments.

The IDLM Grants and Coordinates Resource Lock Requests

This section explains how the IDLM 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 IDLM coordinates lock requests, ensuring compatibility of resource access rights. In this process the IDLM tracks all lock requests. Requests for available resources are granted and access rights granted are tracked. Requests for resources not currently available are tracked, and access rights are granted when these resources later become available. The IDLM inventories these lock requests and communicates their statuses to users and processes involved.

Lock Requests Are Queued

The IDLM maintains two queues for lock requests:

Convert queue	If the IDLM cannot immediately grant a lock request be, it is placed in the convert queue where waiting lock requests are tracked.
Granted queue	The IDLM 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 IDLM uses two types of asynchronous traps (ASTs) or interrupts:

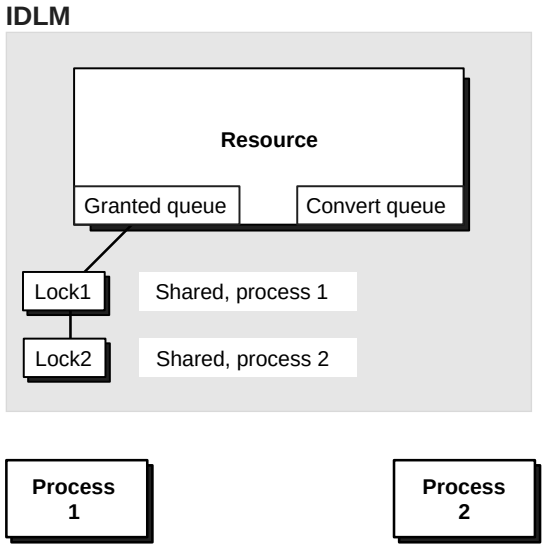
Acquisition AST	When the lock is obtained in the requested mode, an acquisition AST (a "wakeup call") is sent to tell the requestor that the lock is granted.
Blocking AST	When a process requests a certain mode of lock on a resource, the IDLM sends a blocking AST to notify processes currently owning locks 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 IDLM handles lock requests.

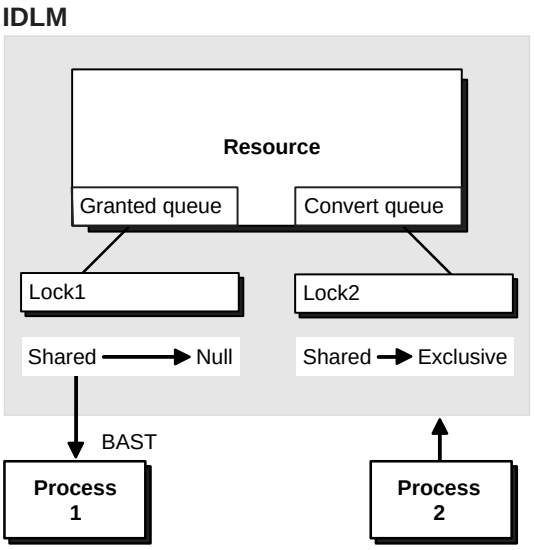
In Figure 8–1, shared lock request 1 has been granted on the resource to process 1, and shared lock request 2 has been granted to process 2. As mentioned, IDLM tracks the locks in the granted queue. When a request for an exclusive lock is made by process 2, it must wait in the convert queue.

Figure 8–1 The IDLM Granted and Convert Queues



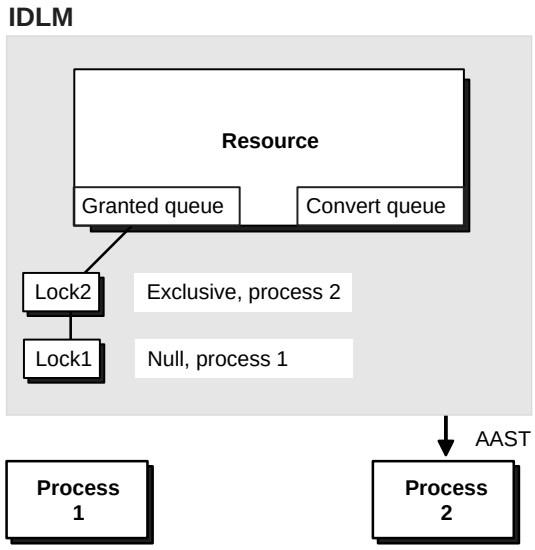
In Figure 8–2, the IDLM sends a blocking AST to Process 1, the owner of the shared lock, notifying it that a request for an exclusive lock is waiting. When the shared lock is relinquished by Process 1, it is converted to a null mode lock or released.

Figure 8–2 Blocking AST



An acquisition AST is then sent to wake up Process 2, the requestor of the exclusive lock. The IDLM grants the exclusive lock and converts it to the granted queue. This is illustrated below in Figure 8–3.

Figure 8–3 Acquisition AST



IDLM Lock Modes: Resource Access Rights

Instances use locks to obtain various rights to a resource. A lock may be initially created on a resource with no access rights granted. Later, a process converts the lock to obtain new access rights.

Figure 8–4 illustrates the levels of access rights or "lock modes" available through the IDLM.

Figure 8–4 IDLM Lock Modes: Levels of Access

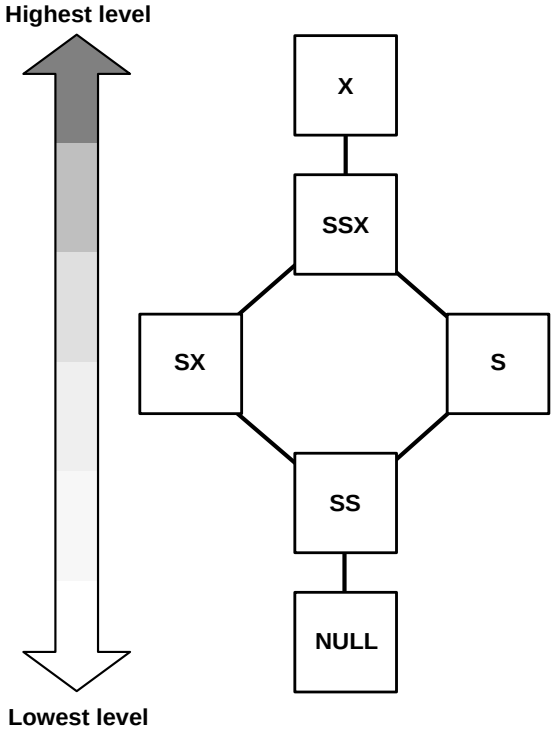


Table 8–1 Lock Mode Names

Oracle 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 IDLM 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.

IDLM Features

This section describes the following features of the IDLM:

- Distributed Architecture
- Fault Tolerance
- Lock Mastering
- Deadlock Detection
- Lamport SCN Generation
- Group-owned Locks
- Persistent Resources
- Memory Requirements
- Support for MTS and XA
- Views to Monitor IDLM Statistics

Distributed Architecture

The IDLM maintains a database of resources and locks held on these resources in different modes. This lock database resides in volatile memory and is distributed.

The IDLM has a distributed architecture. In the distributed architecture each node in the cluster (or each OPS instance of an Oracle database) participates in global lock management and manages a piece of the global lock database. The lock database is distributed among all participants. This distributed lock management scheme provides fault tolerance and enhanced runtime performance.

Fault Tolerance

The IDLM is fault tolerant in that it provides continual service and maintains the integrity of the lock database in the event of multiple node and OPS instance failures. A database is accessible as long as at least one OPS instance is active on that database after recovery completes.

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

Lock Mastering

The IDLM maintains information about locks on all nodes that need access to a particular resource. The IDLM usually nominates one node to manage *all* relevant information about a resource and its locks. This is node called the "master node".

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

The IDLM 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.

When a resource is mastered remotely, all conflicting accesses to this resource result in the transmission of messages to the master node for this resource. This increases internode message traffic and affects system performance.

Deadlock Detection

IDLM performs deadlock detection to all deadlock sensitive locks and resources.

Lamport SCN Generation

OPS uses the fast and scalable Lamport SCN generation scheme that can generate SCNs in parallel on all instances.

See Also: "Lamport SCN Generation" on page 4-7.

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.

See Also: "Support for MTS and XA" on page 8-10.

Persistent Resources

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

See Also: "Lock Requests Are Converted and Granted" on page 8-3.

Memory Requirements

The user-level IDLM 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 IDLM is configured to support all resources your application requires.

Support for MTS and XA

OPS uses two forms of lock ownership:

Per-process ownership	Locks are commonly process-owned: that is, if one process owns a lock exclusively, then no other process can touch the lock.
Group-based ownership	With group-based locking, ownership becomes dynamic: a single lock can be exchanged 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 IDLM grant and convert queues.

Group-based locking is an important IDLM 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.

Views to Monitor IDLM Statistics

Table 8–2 describes six dynamic performance views you can use to monitor IDLM statistics.

Table 8–2 Views to Monitor IDLM Statistics

View	Description
V$DLM_CONVERT_LOCAL	Shows the convert time for local lock convert operations. Enabled by Event 29700.
V$DLM_CONVERT_REMOTE	Shows the convert time for remote lock convert operations. Enabled by event 29700.
V$DLM_LOCKS	Contains debugging information about all locks currently known to the IDLM that are being blocked or are blocking others.
V$DLM_ALL_LOCKS	Contains debugging information about all locks currently known to the IDLM.
V$DLM_RESS	Contains statistics about resources being used by all locks.
V$DLM_MISC	Contains various IDLM statistics.

See Also: Please refer to the *Oracle8 Reference* for a complete description of these dynamic performance views.

9

Parallel Cache Management Instance Locks

The planning and allocation of PCM locks is one of the most complex tasks facing the Oracle Parallel Server (OPS) database administrator. This chapter provides a conceptual overview of PCM locks by covering the following topics:

- PCM Locks and How They Work
- How Initialization Parameters Control Blocks and PCM Locks
- Two Methods of PCM Locking: Fixed and Releasable
- How Oracle Assigns Locks to Blocks
- Examples: Mapping Blocks to PCM Locks

> **See Also:** Chapter 15, "Allocating PCM Instance Locks", for details on how to plan and assign these locks and Chapter 8, "Integrated Distributed Lock Manager" for more information about the IDLM facility.

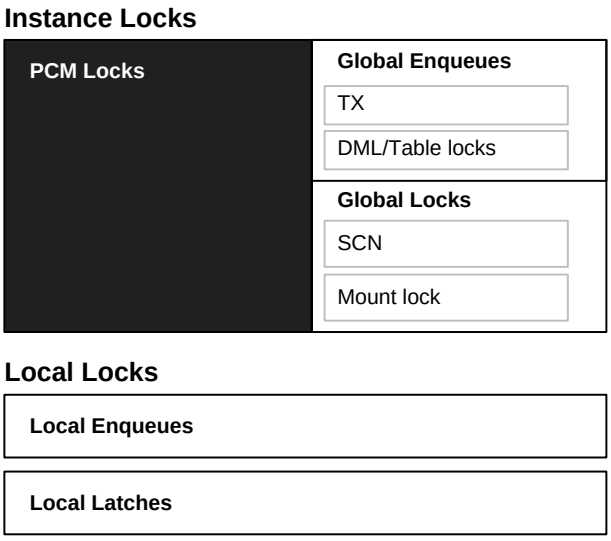
PCM Locks and How They Work

This section covers the following topics:

- What PCM Locks Are
- Allocation and Release of PCM Locks
- How PCM Locks Operate
- Number of Blocks per PCM Lock
- Pinging: Signaling the Need to Update
- Lock Mode and Buffer State

Figure 9–1 highlights PCM locks in relation to other locks used in Oracle.

Figure 9–1 Oracle Locking Mechanisms: PCM Locks



What PCM Locks Are

Parallel cache management locks, or *PCM locks*, are instance locks that manage datafile block locking. PCM locks can cover one or more blocks of any class: data blocks, index blocks, undo blocks, segment headers, and so on. Though there are several types of instance locks, they all serve the same purpose.

OPS uses instance locks to coordinate access to shared resources and the IDLM maintains the statuses of instance locks.

PCM locks ensure cache coherency by forcing requesting instances to acquire locks from holding instances before modifying or reading database blocks. PCM locks allow 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 PCM lock and modify the block.

PCM locks use the minimum amount of communication to ensure cache coherency. The amount of cross-instance activity—and the corresponding performance of OPS—is evaluated in terms of *pings*. A ping occurs each time a block must be written to disk by one instance so that another instance can read it.

Busy systems can have a great deal of locking activity, but do not necessarily have pinging. If data is well partitioned, then the locking will be local to each node—therefore pinging will not occur.

Allocation and Release of PCM Locks

For optimal performance, the OPS administrator must allocate PCM locks to datafiles. You do this by specifying values for initialization parameters which are read at startup of the database. Chapter 15, "Allocating PCM Instance Locks" describes this procedure in detail.

You use the initialization parameter GC_FILES_TO_LOCKS to specify the number of PCM locks which cover the data blocks in a data file or set of data files. The smallest granularity is one PCM lock per datablock; this is the default. PCM locks usually account for the greatest proportion of instance locks in OPS.

Four types of PCM locks can be allocated. They differ in the method by which they are allocated, and in whether or not they are released.

Allocation of Releasable Fine Grain Locks

Fine grain PCM locks are acquired and released as needed. Since they are allocated only as required, the instance can start up much faster than with hashed locks. An IDLM resource is created and an IDLM lock is obtained only when a user actually requests a block. Once a fine grain lock has been created, it can be converted to various modes as required by various instances.

Fine grain locks are releasable: an instance can give up all references to the resource name during normal operation. The IDLM resource 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.

Allocation of Fixed Hashed Locks

Hashed locks are pre-allocated and statically hashed to blocks at startup time. The first instance which starts up creates an IDLM resource and an IDLM lock (in null mode) on the IDLM resource for each hashed PCM lock. The first instance initializes each lock. The instance then proceeds to convert IDLM locks to other modes as required. When a second instance requires a particular IDLM lock, it waits until the lock is available and then converts the lock to the mode required.

By default, hashed PCM locks are never released; each 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.

Allocation of Releasable Hashed Locks

You can specify releasable hashed PCM locks by using the R option with the GC_FILES_TO_LOCKS parameter. Releasable hashed PCM locks are taken from the pool of GC_RELEASABLE_LOCKS.

Allocation of Fixed Fine Grain Locks

You can also allocate fixed locks in a fine grained manner. For example, you could set 50,000 PCM locks for a particular file and thus provide 1 fixed lock for each block.

> **See Also:** "GC_FILES_TO_LOCKS Syntax" on page 15-7 for a detailed explanation of how to set the GC_FILES_TO_LOCKS parameter.

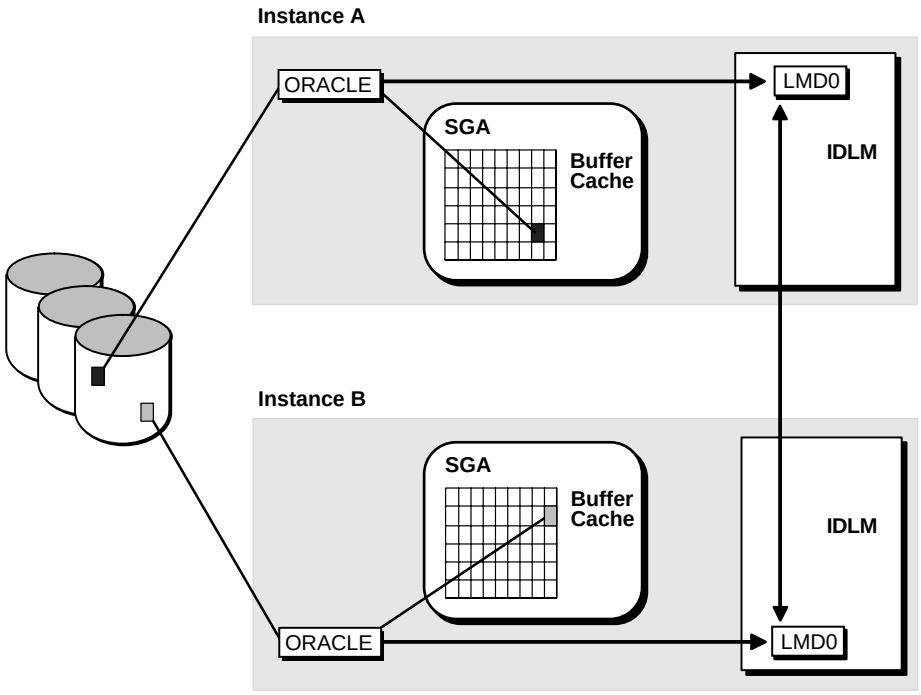
How PCM Locks Operate

Fixed PCM locks are initially acquired in null mode. All specified hashed locks are allocated at instance startup, and de-allocated at instance shutdown. Because of this, hashed locks entail more overhead and longer startup time than fine grain locks. The advantage of fixed hashed PCM locks, however, is that they do not need to be continually acquired and released.

Releasable PCM locking is more dynamic than fixed hashed locking. For example, if you set GC_RELEASABLE_LOCKS to 10000 you can obtain up to ten thousand fine grain PCM locks. However, locks are allocated only as needed by the IDLM. At startup Oracle allocates lock elements that are obtained directly in the requested mode (normally shared or exclusive mode).

Figure 9–2 illustrates how PCM locks work. When instance A reads the black block for modification, it obtains the PCM 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 instance lock from the IDLM.

Figure 9–2 How PCM Locks Work



PCM 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 PCM locks are divided among the LCK processes. This means that each LCK process is only responsible for a subset of the PCM locks.

Locks Convert from One Mode to Another

A PCM lock is "owned" or controlled by an instance when a block covered by that lock (in shared or exclusive mode) enters the buffer cache belonging to the instance.

Multiple Instances Can Own the Same Locks

A PCM lock owned in shared mode is not disowned by an instance if another instance also requests the PCM 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 PCM lock can be contained in the buffer caches of separate instances. This can occur if all the different instances request the PCM lock in shared mode.

Number of Blocks per PCM Lock

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

- If GC_FILES_TO_LOCKS is *not* set for a file, then releasable 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 PCM 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 PCM locks assigned to it, then each hashed PCM lock covers exactly the number of data blocks given by the equation.

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

If *n* files share the same hashed PCM 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.

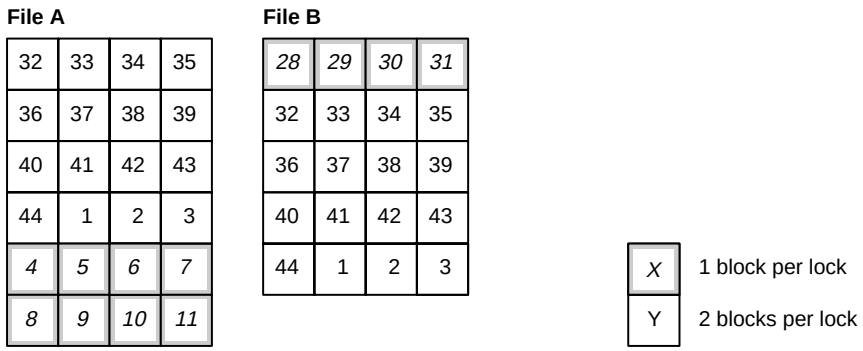
If you assign hashed PCM locks to a set of datafiles collectively, then each lock usually covers one or more blocks in each file. Exceptions can occur when you specify contiguous blocks (using the "!*blocks*" option) or when a file contains fewer blocks than the number of locks assigned to the set of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

Example

The following example illustrates how hashed PCM locks can cover multiple blocks in different files. Figure 9–3 assumes 44 PCM 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 9–3 Hashed PCM Locks Covering Blocks in Multiple Files



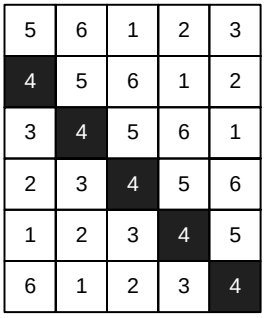
In Figure 9–3, locks 32 through 44 and 1 through 3 are used to cover 2 blocks each. Locks 4 through 11 and 28 through 31 cover 1 block each; and locks 12 through 27 cover no blocks at all!

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 Hashed PCM Locks

You should also consider the *periodicity* of PCM locks. Figure 9–4 shows a file of 30 blocks which is covered by 6 PCM locks. This file has hashed to begin with lock number 5. As suggested by the shaded blocks covered by PCM lock number 4, use of each lock forms a pattern over the blocks of the file.

Figure 9–4 Periodicity of Hashed PCM Locks



Pinging: Signaling the Need to Update

In OPS, 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 the requesting instance submits 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.

BSP (Block Server Process) uses the IDLM 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 unnecessary.

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. If an instance has a block in SHARE mode, it will be pinged if another instance needs it XCUR. In some cases, therefore, the number of PCM locks covering data blocks may have little effect on whether a block gets pinged.

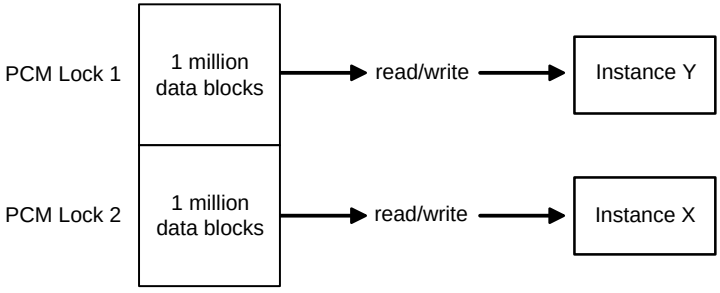
An instance can relinquish an exclusive lock on a block and still have a row lock on it: pinging is independent of whether a commit has occurred. You can modify a block, but whether it is pinged is independent of whether you have made the commit.

Partitioning to Avoid Pinging

If you have partitioned data across instances and are doing updates, you can have a million blocks on each of the different instances. Each block is covered by one PCM lock yet there are no forced reads or forced writes.

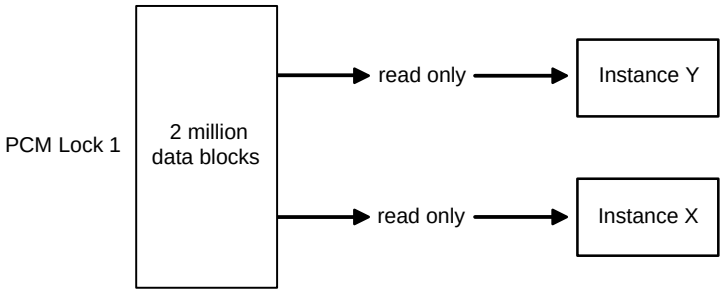
As shown in Figure 9–5, assume a single PCM lock covers one million data blocks in a table and the blocks in that table are read from or written into the SGA of instance X. Assume another single PCM lock covers another million data blocks in the table that are read or written into the SGA of instance 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 9–5 Partitioning Data to Avoid Pinging



With read-only data, both instance X and instance Y can hold the PCM lock in shared mode without causing pinging. This scenario is illustrated in Figure 9–6.

Figure 9–6 No Pinging of Read-only Data



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 PCM 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 learn the state of a buffer, check the STATUS column of the V$BH dynamic performance table. This table provides information about each buffer header.

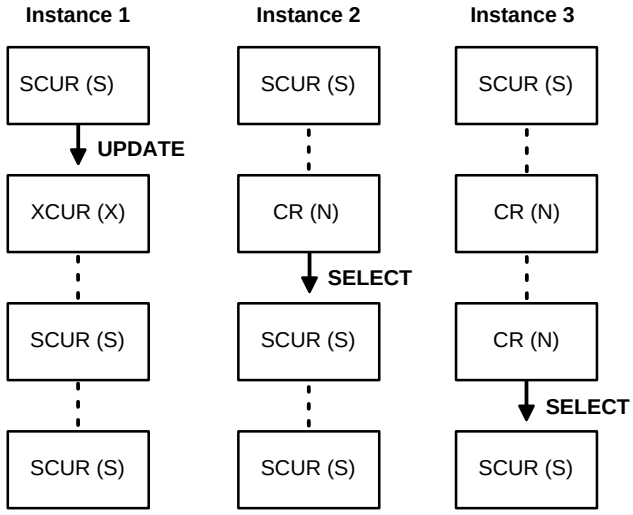
Table 9–1 PCM Lock Mode and Buffer State

PCM 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 State and Lock Mode Change

Figure 9–7 shows how buffer state and lock mode change as instances perform various operations on a given buffer. Lock mode is shown in parentheses.

Figure 9–7 How State of Buffer and Lock Mode Change



In Figure 9–7, 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 becomes CR. These lock modes are compatible. Similar conversions of lock mode and block state occur when Instance 2 performs a SELECT operation on the block, and when Instance 3 performs a SELECT operation on it.

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 9–2.

Table 9–2 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 Initialization Parameters Control Blocks and PCM Locks

This section explains how certain initialization parameters control blocks and PCM locks.

- GC_* Initialization Parameters
- Handling Data Blocks

GC_* Initialization Parameters

PCM locks are controlled by the initialization parameters listed in Table 9–3. Be sure to set *all* of these parameters for your application.

Table 9–3 Parameters Which Control PCM Locks

Parameter	Description	Value
GC_FILES_TO_LOCKS	Gives the mapping of hashed and releasable locks to blocks within each datafile. The meaning of this parameter has changed. Previously, files not mentioned in this parameter (or files added later) were assigned the remaining hashed locks. Files not mentioned in this parameter use releasable locks. You can now have multiple entries of GC_FILES_TO_LOCKS.	The configuration string for GC_FILES_TO_LOCKS now includes a value of zero for the number of locks. This indicates that the blocks are protected by fine grain locks. Instances must have identical values.
GC_RELEASABLE_LOCKS	Sets the number of locks which will be used for DBA locks.	Defaults to the value of DB_BLOCK_BUFFERS. Normally this value is optimal, and you should not change it. In versions prior to Oracle8, setting this parameter to a value less than DB_BLOCK_BUFFERS was ineffective: the value was automatically returned to this default. In Oracle8, lower settings are valid. If you have migrated from an earlier version, check the setting of this parameter to avoid negative effects on performance.
GC_ROLLBACK_LOCKS	For each rollback segment, specifies the number of instance locks available for simultaneously modified rollback segment blocks.	The default value is to use releasable locks for each rollback segment.

See Also: *Oracle8i Reference* for complete specifications for these parameters and Chapter 15, "Allocating PCM Instance Locks", provides information on how to set these parameters.

Handling Data Blocks

Do *not* allocate PCM locks for files that *only* contain the following, because class 1 blocks are not used for these files:

- Temporary tables for internal sorts. These are class 2 blocks.
- Rollback segments. These are protected by GC_ROLLBACK_LOCKS.

Two Methods of PCM Locking: Fixed and Releasable

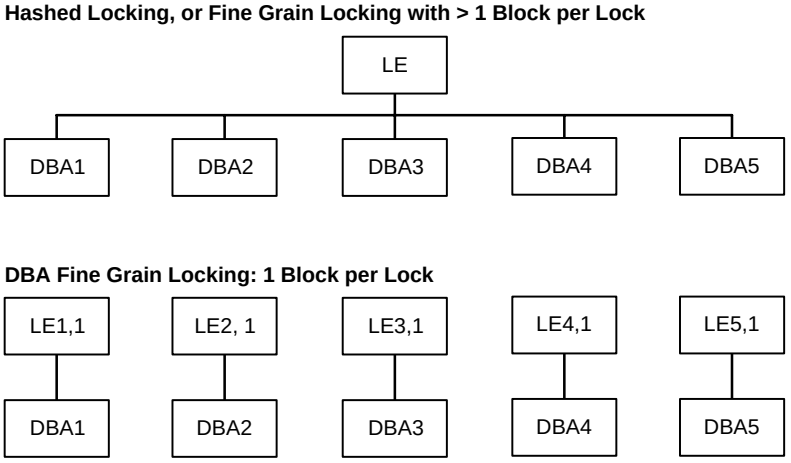
This section compares the two methods for PCM locking: fixed (hashed) and releasable locking. You can use either or both types of PCM locks to protect datafile blocks.

- IDLM Lock Elements and PCM Locks
- Number of Blocks per PCM Lock
- Fine Grain Locking: Locks for One or More Blocks
- How Fine Grain Locking Works
- Performance Effects of Releasable Locking
- Applying Fine Grain and Hashed Locking to Different Files

IDLM Lock Elements and PCM Locks

Figure 9–8 illustrates the correspondence of lock elements to blocks in hashed and fine grain locking. A lock element (LE) is an Oracle-specific data structure that represents an IDLM lock. There is a one-to-one correspondence between a lock element and a PCM lock in the IDLM.

Figure 9–8 Hashed Locking and DBA Fine Grain Locking



Lock Elements for Fixed PCM Locks

For both fixed PCM locks and fine grain locks, you can specify more than 1 block per lock element. The difference is that by default fixed PCM locks are not releasable; the lock element name is "fixed".

When the lock element is pinged, other modified blocks owned by that lock element are written along with the needed one. For example, in Figure 9–8, if LE is pinged when block DBA2 is needed, blocks DBA1, DBA3, DBA4, and DBA5 are all written to disk as well—if they have been modified.

Lock Elements for Fine Grain PCM Locks

In fine grain locking, the name of the lock element is the name of the resource inside the IDLM.

Although a fixed number of lock elements cover potentially millions of blocks, the lock element names change over and over again as they are associated with specific 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 IDLM lock must be released. You can then rename and reuse the lock element, creating a new resource in the IDLM if necessary.

When using fine grain locking, you can set 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 hashed locking.

Lock Elements for DBA Fine Grain PCM Locks

In fine grain locking you can set a one-to-one relationship between lock elements and blocks. Such an arrangement, illustrated in Figure 9–8, is called *DBA locking*. Thus if LE2,1 is pinged, only block DBA2 is written to disk.

Number of Blocks per PCM Lock

This section explains the ways in which hashed locks and fine grain locks can differ in lock granularity.

Fixed Locks for Multiple Blocks

Fixed PCM locks can protect more than one Oracle database block. The mapping of PCM locks to blocks in the database is determined on a file-by-file basis using initialization parameters specified when the first OPS instance is started. The parameters can specify that the PCM lock protects a range of contiguous blocks within the file.

Hashed locks are useful in the following situations:

Table 9–4 When to Use Hashed PCM Locks

Situation	Reason
When the data is mostly read-only.	A few hashed locks can cover many blocks without requiring frequent lock operations. These locks are released only when another instance needs to modify the data. Hashed locking can perform up to 100% faster than fine grain 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 PCM locks.
When the data can be partitioned according to the instance which is likely to modify it.	Hashed locks which are defined to match this partitioning allow instances to hold disjoint IDLM lock sets, reducing the need for IDLM operations.
When a large amount of data is modified by a relatively small set of instances.	Hashed locks permit access to a new database block to proceed without IDLM activity, if the lock is already held by the requesting instance.

Hashed locks may cause extra cross-instance lock activity since conflicts may occur between instances that modify different database blocks. Resolution of this false conflict ("false pinging") may require writing several blocks from the cache of the instance that currently owns the lock.

Fine Grain Locking: Locks for One or More Blocks

A fine grain lock can protect one or more Oracle database blocks. If you create a one-to-one correspondence between PCM locks and datablocks, then contention will occur only when instances need data from the same block. This level of fine grain locking is known as DBA locking. (A *DBA* is the data block address of a single block of data.) If you assign more than one block per lock, then contention will occur as in hashed locking.

On most systems an instance could not possibly hold a lock for each block of a database since SGA memory or IDLM locking capabilities would be exceeded. Therefore, instances acquire and release fine grain locks as required. Since fine grain locks, lock elements, and resources are renamed in the IDLM and reused, a system can employ fewer of them. The value of DB_BLOCK_BUFFERS is the recommended minimum number of releasable locks you should allocate.

DBA fine grain locks are useful when a database object is updated frequently by several instances. This advantage is gained as follows:

- 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 PCM lock *in exclusive mode*

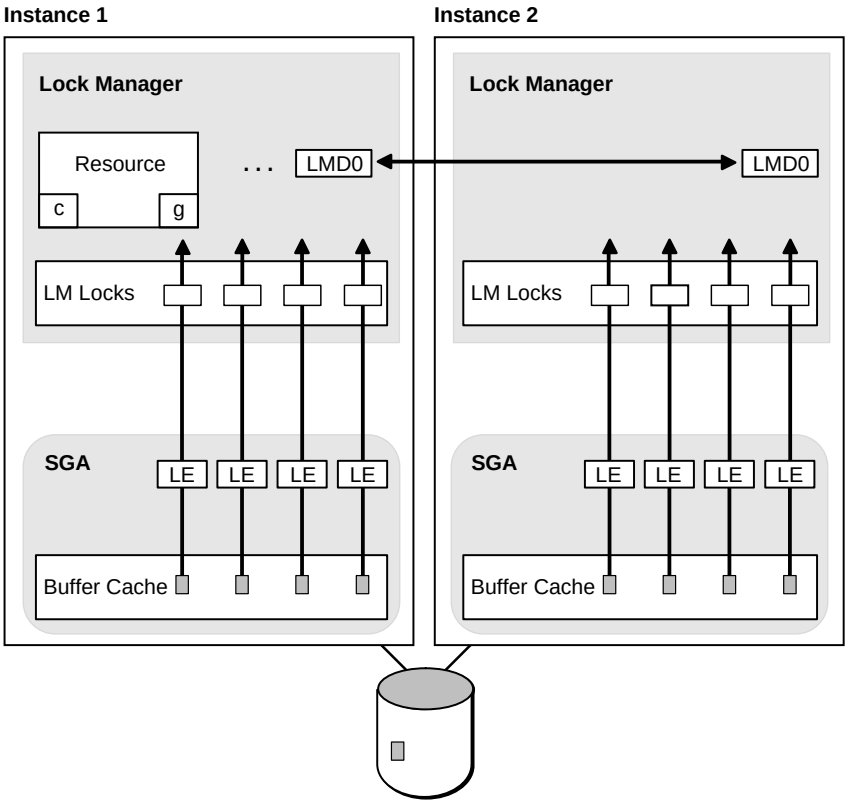
A disadvantage of fine grain locking is that overhead is incurred for each block read, and performance will be affected accordingly. (Acquiring a new lock and releasing it each time causes more overhead through the IDLM than converting the mode of an existing lock from null mode to exclusive mode and back, as is done in hashed locking.)

See Also: "Releasable Lock Example" on page 15-9.

How Fine Grain Locking Works

Figure 9–9 shows how fine grain locking operates.

Figure 9–9 Lock Elements Coordinate Blocks (by Fine Grain Locking)



The foreground process checks in the SGA to see if the instance owns a lock on the block.

- If the lock is not owned does not exist, the foreground process creates one by releasing another lock.
- If the instance does own the lock, but in the wrong mode, then the foreground process converts the lock (for example, from shared to exclusive mode).

A lock element is created in either of two modes: fixed or releasable

- Fixed locks (whether hashed or fine grain) create a lock element in fixed mode, which is always valid. This mode is static; lock elements stay the same, once allocated.
- Releasable locks (whether fine grain or hashed) create a lock element in non-fixed 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 IDLM. 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 IDLM; invalid lock elements do not. A free lock element indicates that a lock exists in the IDLM which is not currently linked to this buffer; it is waiting on the LRU 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.

The V$LOCK_ELEMENT view shows the status of the lock elements.

Performance Effects of Releasable Locking

Releasable locking may affect performance of OPS. Since releasable locks are more expensive (since they may cause a release lock and get lock on a buffer get), some operations may show a decreased level of performance when run in this mode. However, other types of access to the database will improve with releasable fine grain locks. Fine grain locking may have the following results:

- Read-only scans of tables may require more lock operations. If this happens, you can use hashed locking.
- There may be a reduction of false conflicts on high-contention blocks or objects. If this happens, fine grain locking is a good choice.
- The system has more expensive lock operations and a lower false conflict rate for low concurrency data blocks. If this happens you must examine your priorities and decide whether this is a reasonable trade-off for your application.
- Grouping with fine grain locks might be good for table scans.

Applying Fine Grain and Hashed Locking to Different Files

Each datafile can use one or the other method of locking. For best results, you may need to use hashed locks on some datafiles, and fine grain locking on other datafiles.

You can selectively apply hashed and fine grain locking on different files. For example, you could apply locks as follows on a set of files:

GC_FILES_TO_LOCKS = "1=100:2=0:3=1000:4-5=0EACH"
GC_RELEASABLE_LOCKS=10000

Table 9–5 Selective Application of Hashed and Fine Grain Locking

File Number	Locking Mode	Value in GC_FILES_TO_LOCKS
1	Hashed	100
2	Fine grain	0
3	Hashed	1000
4	Fine grain	0
5	Fine grain	0

How Oracle Assigns Locks to Blocks

This section explains how hashed locks and fine grain locks are assigned to blocks. (DBA locks, of course, have a one-to-one correspondence to blocks.)

- File to Lock Mapping
- Number of Locks per Block Class
- Lock Element Number

File to Lock Mapping

Two data structures in the SGA control file to lock mapping. The first structure maps each file (DB_FILES) to a bucket (index) in the second structure. This structure contains information on the number of locks allocated to this bucket, base lock number and grouping factor. To find the number of locks for a tablespace, you must count the number of actual fixed locks which protect the different files. If files share locks, you count the shared locks only once.

1. To find the number of locks for a tablespace, begin by performing a select from the FILE_LOCK data dictionary table:

   ```
   SELECT * FROM FILE_LOCK ORDER BY FILE_ID;
   ```

   For example, Oracle would respond with something similar to the following if you had set GC_FILES_TO_LOCKS="1=500:5=200":

   ```
   FILE_ID FILE_NAME       TS_NAME          START_LK     NLOCKS   BLOCKING
   ------- --------------- -------------- ---------- ---------- ----------
         1 \\.\OPS_SYS01   SYSTEM                100       1500          1
         2 \\.\OPS_USR01   USER_DATA            1600       3000          1
         3 \\.\OPS_RBS01   ROLLBACK_DATA           0        100          1
         4 \\.\OPS_TMP01   TEMPORARY_DATA         0        100          1
         5 \\.\OPS_USR03   TRAVEL_DEMO         4600       4000          1
         6 \\.\PROBLEM_REP PROBLEM_REP           0        100          1

   6 rows selected.
   ```

2. Count the number of locks in the tablespace by summing the number of locks (value of the NLOCKS column) *only for rows with different values in the START_LCK column*.

   In this example, both file1 and file5 have different values for START_LCK. You therefore sum their NLOCKS values for a total of 700 locks.

If, however, you had set GC_FILES_TO_LOCKS="1-2=500:5=200", your results would look like the following:

```
FILE_ID  FILE_NAME  TABLESPACE_NAME  START_LK  NLOCKS BLOCKING
1        file1      system             1         500    1
1        file2      system             1         500    1
1        file3      system             0
1        file4      system             0
1        file5      system           501         200    1
```

This time, file1 and file 2 have the same value for START_LCK; this indicates that they share the locks in question. File5 has a different value for START_LCK. You therefore count once the 500 locks shared by files 1 and 2, and add an additional 200 locks for file 5, for a total of 700.

Number of Locks per Block Class

You need only concern yourself with the number of blocks in the data and undo block classes. Data blocks (class 1) contain data from indexes or tables. System undo header blocks (class 10) are also known as the rollback segment headers or transaction tables. System undo blocks (class 11) are part of the rollback segment and provide storage for undo records.

User undo segment *n* header blocks are identified as class 10 + (*n**2), where *n* represents the rollback segment number. A value of *n* = 0 indicates a system rollback segment; a value of *n* > 0 indicates a non-system rollback segment. Similarly, user undo segment *n* header blocks are identified as class 10 + ((n*2) + 1).

The following query shows the number of locks allocated per class:

```
SELECT CLASS, COUNT(*)
  FROM V$LOCK_ELEMENT
  GROUP BY CLASS
  ORDER BY CLASS;
```

The following query shows the number of fixed (non-releasable) PCM locks:

```
SELECT COUNT(*)
  FROM V$LOCK_ELEMENT
  WHERE bitand(flag, 4)!=0;
```

The following query shows the number of fine grain PCM locks:

```
SELECT COUNT(*)
  FROM V$LOCK_ELEMENT
  WHERE bitand(flag, 4)=0;
```

Lock Element Number

For a data class block the file number is determined from the data block address (DBA). The bucket is found through the X$KCLFI dynamic performance table. Data class blocks are hashed to lock element numbers as follows:

$$\left(\frac{DBA}{grouping\_factor} \right) \text{ modulo } (locks) + (start)$$

Other block classes are hashed to lock element numbers as follows:

(*DBA*) modulo (*locks_in_class*)

Examples: Mapping Blocks to PCM Locks

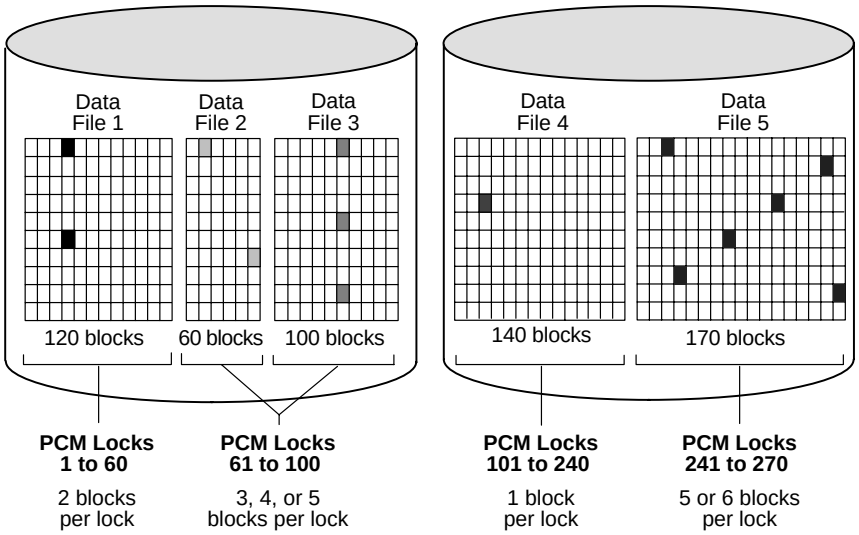
- Setting GC_FILES_ TO_LOCKS
- More Sample Hashed Settings of GC_FILES_TO_LOCKS
- Sample Fine Grain Setting of GC_FILES_TO_LOCKS

Setting GC_FILES_ TO_LOCKS

The following examples show different ways of mapping blocks to PCM locks, and how the same locks are used on multiple datafiles.

Note: These examples discuss very small sample files to illustrate important concepts. The actual files you manage will be significantly larger.

Figure 9–10 Mapping PCM Locks to Data Blocks



Example 1 Figure 9–10 shows an example of mapping blocks to PCM locks for the parameter value GC_FILES_TO_LOCKS = "1=60:2-3=40:4=140:5=30".

In datafile 1 shown in Figure 9–10, 60 PCM locks map to 120 blocks, which is a multiple of 60. Each PCM lock therefore covers two data blocks.

In datafiles 2 and 3, 40 PCM locks map to a total of 160 blocks. A PCM lock can cover either one or two data blocks in datafile 2, and two or three data blocks in datafile 3. Thus, one PCM lock may cover three, four, or five data blocks across both datafiles.

In datafile 4, each PCM lock maps exactly to a single data block, since there is the same number of PCM locks as data blocks.

In datafile 5, 30 PCM locks map to 170 blocks, which is not a multiple of 30. Each PCM lock therefore covers five or six data blocks.

Each of the PCM locks illustrated in Figure 9–10 can be held in either read-lock mode or read-exclusive mode.

Example 2 The following parameter value allocates 500 PCM locks to datafile 1; 400 PCM locks each to files 2, 3, 4, 10, 11, and 12; 150 PCM locks to file 5; 250 PCM locks to file 6; and 300 PCM locks collectively to files 7 through 9:

```
GC_FILES_TO_LOCKS = "1=500:2-4,10-12=400EACH:5=150:6=250:7-9=300"
```

This example assigns a total of (500 + (6*400) + 150 + 250 + 300) = 3600 PCM locks. You may specify more than this number of PCM locks if you intend to add more datafiles later.

Example 3 In Example 2, 300 PCM locks are allocated to datafiles 7, 8, and 9 collectively with the clause "7-9=300". The keyword EACH is omitted. If each of these datafiles contains 900 data blocks, for a total of 2700 data blocks, then each PCM lock covers 9 data blocks. Because the datafiles are multiples of 300, the 9 data blocks covered by the PCM lock are spread across the 3 datafiles; that is, one PCM lock covers 3 data blocks in each datafile.

Example 4 The following parameter value allocates 200 PCM locks each to files 1 through 3; 50 PCM locks to datafile 4; 100 PCM locks collectively to datafiles 5, 6, 7, and 9; and 20 data locks in contiguous 50-block groups to datafiles 8 and 10 combined:

```
GC_FILES_TO_LOCKS = "1-3=200EACH 4=50:5-7,9=100:8,10=20!50"
```

In this example, a PCM lock assigned to the combined datafiles 5, 6, 7, and 9 covers one or more data blocks in each datafile, unless a datafile contains fewer than 100 data blocks. If datafiles 5 to 7 contain 500 data blocks each and datafile 9 contains 100 data blocks, then each PCM lock covers 16 data blocks: one in datafile 9 and five each in the other datafiles. Alternatively, if datafile 9 contained 50 data blocks, half of the PCM locks would cover 16 data blocks (one in datafile 9); the other half of the PCM locks would only cover 15 data blocks (none in datafile 9).

The 20 PCM locks assigned collectively to datafiles 8 and 10 cover contiguous groups of 50 data blocks. If the datafiles contain multiples of 50 data blocks and the total number of data blocks is not greater than 20 times 50 (that is, 1000), then each PCM lock covers data blocks in either datafile 8 or datafile 10, but not in both. This is because each of these PCM locks covers 50 contiguous data blocks. If the size of datafile 8 is not a multiple of 50 data blocks, then one PCM lock must cover data blocks in both files. If the sizes of datafiles 8 and 10 exceed 1000 data blocks, then some PCM locks must cover more than one group of 50 data blocks, and the groups might be in different files.

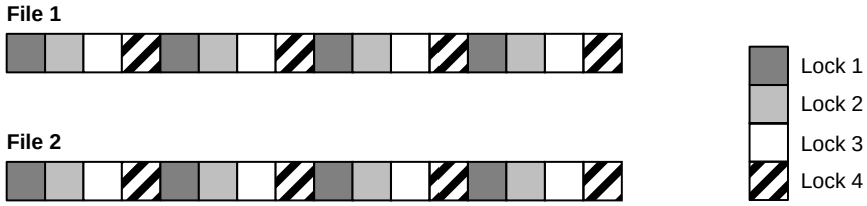
More Sample Hashed Settings of GC_FILES_TO_LOCKS

Examples 5, 6, and 7 show the results of specifying various values of GC_FILES_TO_LOCKS. In the examples, files 1 and 2 each have 16 blocks of data.

Example 5 GC_FILES_TO_LOCKS="1-2=4"

In this example four locks are specified for files 1 and 2. Therefore, the number of blocks covered by each lock is 8 ((16+16)/4). The blocks are not contiguous.

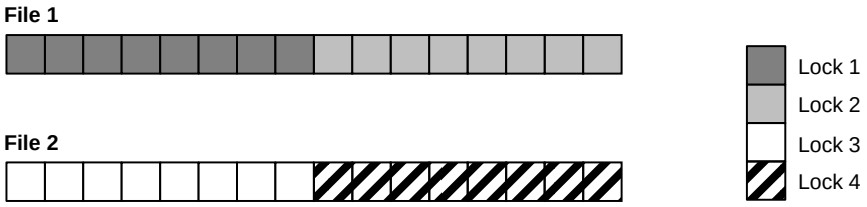
Figure 9–11 GC_FILES_TO_LOCKS Example 5



Example 6 GC_FILES_TO_LOCKS="1-2=4!8"

In this example four locks are specified for files 1 and 2. However, the locks must cover 8 contiguous blocks.

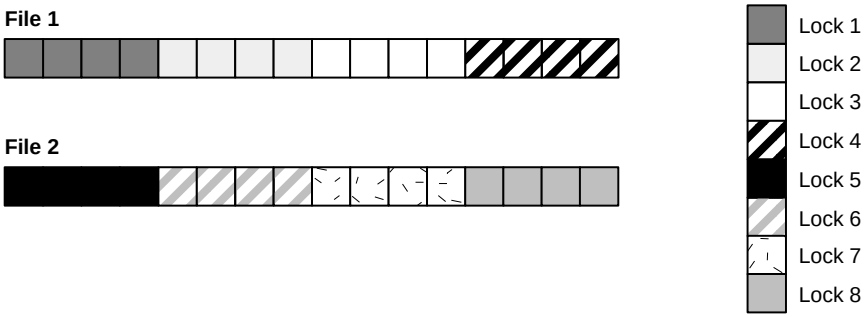
Figure 9–12 GC_FILES_TO_LOCKS Example 6



Example 7 GC_FILES_TO_LOCKS="1-2=4!4EACH"

In this example four locks are specified for file 1 and four for file 2. The locks must cover 4 contiguous blocks.

Figure 9–13 GC_FILES_TO_LOCKS Example 7



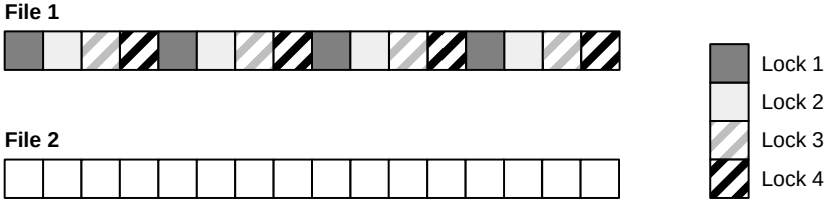
Sample Fine Grain Setting of GC_FILES_TO_LOCKS

The following example shows fine grain locking mixed with hashed locking.

Example 8 GC_FILES_TO_LOCKS="1=4:2=0"

File 1 has hashed PCM locking with 4 locks. On file 2, fine grain locks are allocated on demand—none are initially allocated.

Figure 9–14 GC_FILES_TO_LOCKS Example 8



10

Non-PCM Instance Locks

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

- Overview
- Transaction Locks (TX)
- Table Locks (TM)
- System Change Number (SC)
- Library Cache Locks (N[A-Z])
- Dictionary Cache Locks (Q[A-Z])
- Database Mount Lock (DM)

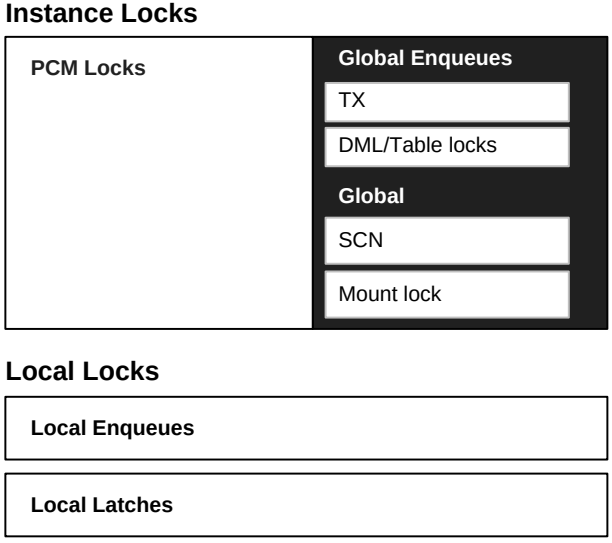
> **See Also:** Chapter 16, "Ensuring IDLM Capacity for Resources and Locks", for details on how to calculate the number of non-PCM resources and locks to configure in the Integrated Distributed Lock Manager (IDLM).

Overview

This chapter explains how Oracle uses non-PCM locks to manage locks for transactions, tables, and other entities within an Oracle environment. Prefixes for each type of lock, such as "TX" for transaction locks and "TM" for table locks, refer to the naming scheme Oracle uses to identify them.

Figure 10–1 highlights non-PCM locks in relation to other locks used in Oracle.

Figure 10–1 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 (TX)

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 (undo) 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 (TM)

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).

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

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

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

If DML_LOCKS is set to zero, then no DDL operations are allowed. The same is true for tables which have disabled table locks.

> **See Also:** "Minimizing Table Locks to Optimize Performance" on page 16-6 to consider disabling table locks for improved performance.

System Change Number (SC)

The System Change Number (SCN) is a logical timestamp Oracle uses to order events within a single instance, and across all instances. One of the schemes Oracle uses to generate SCNs is the lock scheme.

The lock SCN scheme keeps the global SCN in the value block of the SCN lock. This value is incremented in response to many database events, most notably COMMIT WORK. A process incrementing the global SCN will get the SCN lock in exclusive mode, increment the SCN, write the lock value block, and downgrade the lock. Access to the SCN lock value is batched. Oracle keeps a cache copy of the global SCN in memory. A process may get an SCN without any communication overhead by reading the SCN fetched by other processes.

The SCN implementation can differ from platform to platform. On most platforms, Oracle uses the lock SCN scheme when the MAX_COMMIT_PROPAGATION_DELAY initialization parameter is smaller than a platform-specific threshold (typically 7). Oracle uses the Lamport SCN scheme when MAX_COMMIT_PROPAGATION_DELAY is larger than the threshold.You can examine the alert log after an instance is started to see which SCN generation scheme has been picked.

> **See Also:** Your Oracle system-specific documentation for information about the SCN implementation.

Library Cache Locks (N[A-Z])

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) or PL / SQL 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 (Q[A-Z])

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. Instance locks are used in multiple shared (parallel) mode.

In Oracle Parallel Server (OPS), 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 instance locks.

Database Mount Lock (DM)

The mount lock shows whether or not any instance has mounted a particular database. This lock is only used with OPS. It is the only multi-instance lock used by OPS in exclusive mode, where it prevents another instance from mounting the database in shared mode.

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

11

Space Management and Free List Groups

Thus would I double my life's fading space;
For he that runs it well, runs twice his race.

Abraham Cowley, *Discourse xi, Of Myself*

This chapter explains space management concepts:

- How Oracle Handles Free Space
- SQL Options for Managing Free Space
- Managing Free Space on Multiple Instances
- Free Lists Associated with Instances, Users, and Locks
- Controlling Extent Allocation

> **See Also:** Chapter 17, "Using Free List Groups to Partition Data", for a description of space management procedures.

How Oracle Handles Free Space

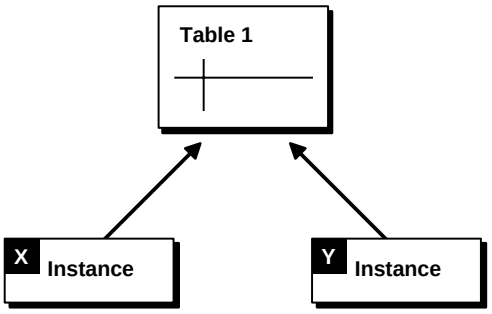
This section provides an overview of how Oracle handles free space. It contains the following sections:

- Overview
- Database Storage Structures
- Structures for Managing Free Space
- Example: Free List Groups

Overview

Oracle Parallel Server (OPS) enables transactions running on separate instances to insert and update data in the same table concurrently, without contention to locate free space for new records.

Figure 11–1 Instances Concurrently Inserting to a Table



To take advantage of this capability, you must actively manage free space in your database using several structures which are defined in this chapter.

For each database object such as a table, cluster, or index, Oracle keeps track of blocks with space available for inserts, or for updates which may cause rows to exceed the space available in their original block. A user process that needs free space can look in the master free list of blocks that contain free space. If the master free list does not contain a block with enough space to accommodate the user process, Oracle allocates a new extent.

New extents that are automatically allocated to a table add their blocks to the master free list. This can eventually result in contention for free space among multiple instances on a parallel server because the free space contained in automatically 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 can minimize contention for free space.

Database Storage Structures

This section describes basic structures of database storage:

- Segments and Extents
- High Water Mark

Segments and Extents

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.

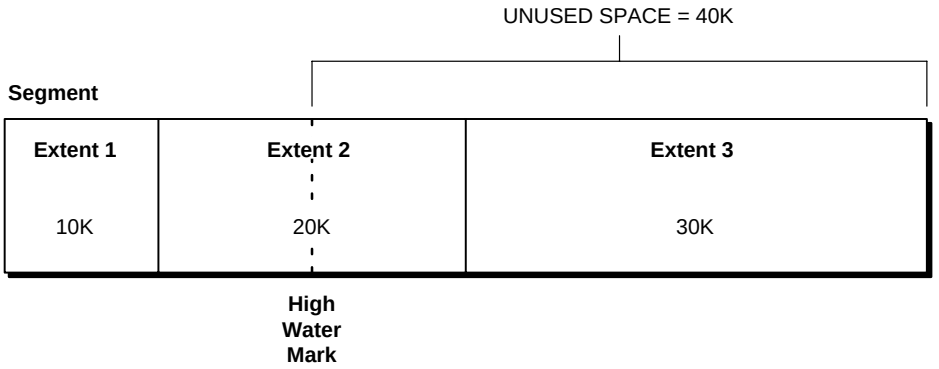
Although you can allocate additional extents, the blocks themselves are allocated separately. If you allocate an extent to a specific instance, the blocks are immediately allocated to the free list. However, if the extent is not allocated to a specific instance, then the blocks themselves are allocated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

High Water Mark

The *high water mark* is the boundary between used and unused space in a segment. As requests are received for new free blocks (which cannot be satisfied by existing free lists), 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 11–2 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 11–2 High Water Mark



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

Structures for Managing Free Space

Oracle uses the following structures to manage free space:

- Transaction Free Lists
- Process Free Lists
- Free List Groups
- The Master Free List

Process free lists relieve contention for free space among processes inside the instance, even if multiple instances can hash to a single free list group. Free list groups relieve forced reads / writes between instances. Process free lists and free list groups are supported on *all* database objects alike: tables, indexes, and clusters.

Transaction Free Lists

A *transaction free list* is a list of blocks freed by uncommitted transactions. They exist by default. When transactions are committed, the freed blocks eventually go to the master free list as described under the following heading.

Process Free Lists

A *process free list*, also termed simply a "free list", is a list of free data blocks that can be drawn from a number of different extents within the segment.

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 described in the chapter "Data Blocks, Extents, and Segments" in *Oracle8i Concepts*.

Process free lists must be specifically enabled by the user. You can specify the number of process free lists desired 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 handle overhead.

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 you can specify for use by one or more particular 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.

A parallel server has multiple instances, and process free lists alone cannot solve the problem of contention. Free list groups, however, effectively reduce pinging between instances.

When enabled, free list groups divide the set of free lists into subsets. Descriptions of process free lists are stored in separate blocks for the different free list groups. Each free list group block points to the same free lists, except that every instance gets its own. (Or, in the case of more instances than free list groups, multiple instances hash into the same free list group.) This ensures that the instances do not compete for the same blocks.

Note: In OPS, always use free list groups along with process free lists.

The Master Free List

The master free list is a repository of blocks which contain available space, drawn from any extent in the table. It exists by default, and includes:

- Blocks which were made free by a committed transaction. These go on the master free list when there is a need for free blocks.
- Subsequent space allocations not specifically associated with any free list group. When the high water mark moves, then blocks go on the master free list.

If free list groups exist, each group has its own master free list. There is, in addition, a central master free list which is mostly used for parallel operations.

Avoiding Contention for the Segment Header and Master Free LIst

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

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

In a single instance environment, multiple process free lists help to solve the problem of many users seeking free data blocks by easing contention on segment header blocks.

In a multi-instance environment, as illustrated in Figure 11–3, process 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 process free lists.

If no free list groups are allocated, however, the segment header block of a file points to the process free lists. Without free list groups, every instance must read the segment header block in order to access the free lists.

Figure 11–3 Contention for the Segment Header

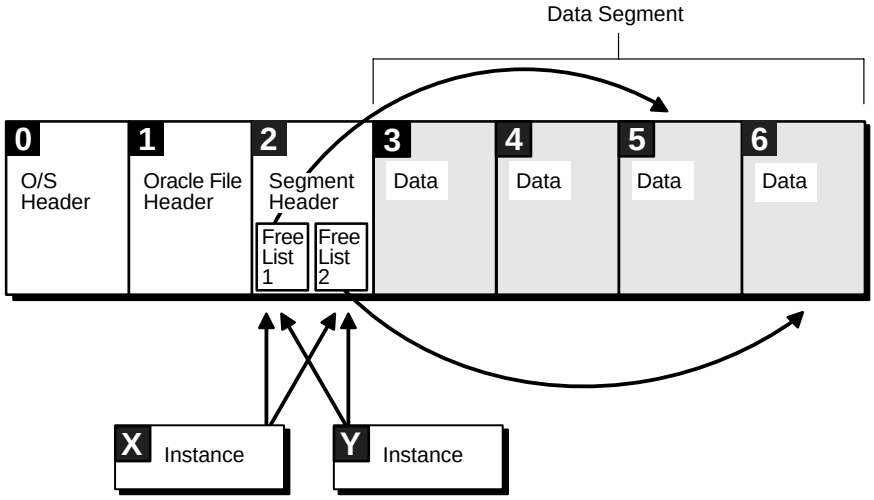
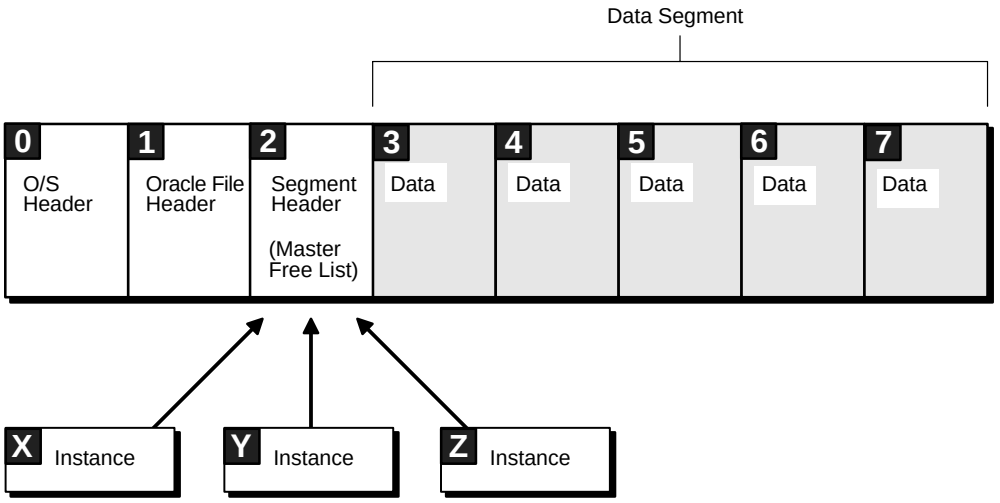


Figure 11–4 shows the blocks of a file in which the master free list is stored in the segment header block. Three instances are forced to read this block to obtain free space. Because there is only one free list, there is only one insertion point. Process free lists help reduce contention by spreading this insertion point over multiple blocks, each of which will be accessed less often.

Figure 11–4 Contention for Master Free List



Example: Free List Groups

A Simple Case

Figure 11–5 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 concerns 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 and extent 6 represent the master free list of free blocks.
- The light gray blocks represent available free space in free list group X.
- The medium gray blocks 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.
- Solid black blocks represent space freed by deletions, which returns to free list groups X and Y.

- Unshaded blocks do not contain enough free space for inserts.

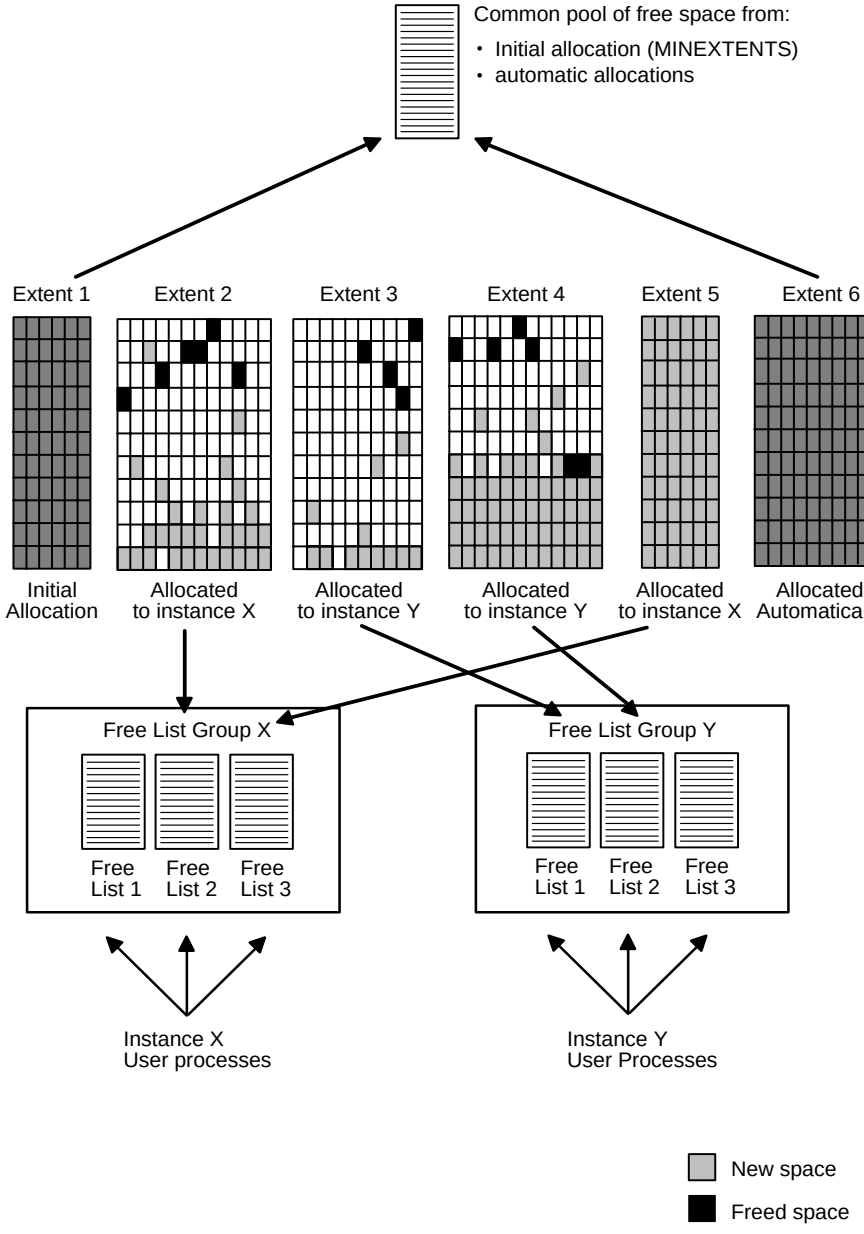
Each user process running on instance X uses one of the free lists in group X, and each user process on instance Y uses one of the free lists in group Y. If more instances start up, their user processes share free lists with instance X or Y.

A More Complicated Case

The simple case in Figure 11–5 becomes more complicated when you consider that extents are not allocated to instances permanently, 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 11–5 Groups of Free Lists for a Table



SQL Options for Managing Free Space

Several SQL options enable you to allocate process 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 datafile. You can also associate free space with particular free list groups, that can then be associated 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.

Managing Free Space on Multiple Instances

This section describes:

- Partitioning Free Space into Multiple Free Lists
- Partitioning Data with Free List Groups
- How Free Lists and Free List Groups Are Assigned to Instances

Partitioning Free Space into Multiple Free Lists

You can partition free space for individual tables, clusters (other than hash clusters), and indexes into multiple process free lists. Multiple free lists allow a process to search a specific pool of blocks when space is needed, thus reducing contention among users for free space. Within an instance, using free lists can reduce contention if multiple processes are inserting into the same table.

Each table has a master free list of blocks with available space, and can also contain multiple free lists. Before looking in the master free list, a user process scans the appropriate free list to locate a block that contains enough space.

Partitioning Data with Free List Groups

The separation of free space into groups can improve performance by reducing contention for free data blocks during concurrent inserting by multiple instances on OPS. You can thus create groups of process free lists for OPS, each of which can contain multiple free lists for a table, index, or cluster. You can use free list groups to partition data by allocating extents to particular instances.

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.

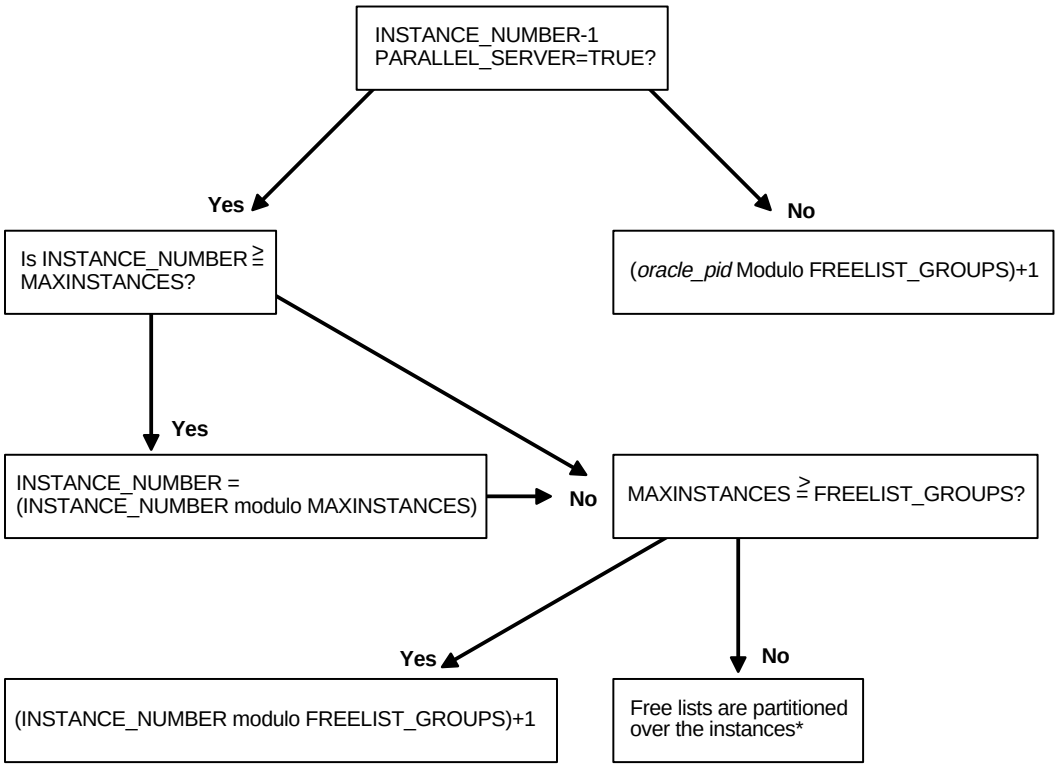
In a multi-instance environment, information about multiple free lists and free list groups is not preserved upon import. If you use Export and Import to back up and restore your data, it will be difficult to import the data so that it is partitioned again.

> **See Also:** "Free Lists with Import and Export Utilities" and Chapter 12, "Application Analysis" for more information on partitioning data. Also see *Oracle8i Tuning*.

How Free Lists and Free List Groups Are Assigned to Instances

Figure 11–6 illustrates how free lists and free list groups are assigned to instances.

Figure 11–6 How Free Lists and Free List Groups Are Assigned



Using the statement ALTER SESSION INSTANCE_NUMBER, you can alter the instance number to be larger than the value of MAXINSTANCES. Figure 11–6 shows how this possibility 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.

* Free lists are partitioned as follows: If there are 3 instances and 35 free list groups, then instance 1 will handle the first twelve free list groups, instance 2 the next twelve, and instance 3 the remaining eleven. The actual free list group block is determined by hashing *oracle_pid* by the number of free list groups.

Free Lists Associated with Instances, Users, and Locks

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

A table can have separate groups of process free lists assigned to particular instances. Each group of free lists can be associated with a single instance, or several instances can share one group of free lists. All instances also have access to the master free list of available space.

Groups of free lists allow you to associate instances with different sets of data blocks for concurrent inserts and updates requiring new space. This reduces contention for the segment header block, which contains information about the master free list of free blocks. For tables that do not have multiple free list groups, the segment header also contains information about free lists for user processes. You can use free list groups to locate the data that an instance inserts and accesses frequently in extents allocated to that instance.

Data partitioning can reduce contention for data blocks. Often the PCM locks that 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, such as a clustered system, 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 system, however, MAXINSTANCES could be many times larger than FREELIST GROUPS so that many instances share one group of free lists.

See Also: "Associating Instances, Users, and Locks with Free List Groups" on page 17-9.

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

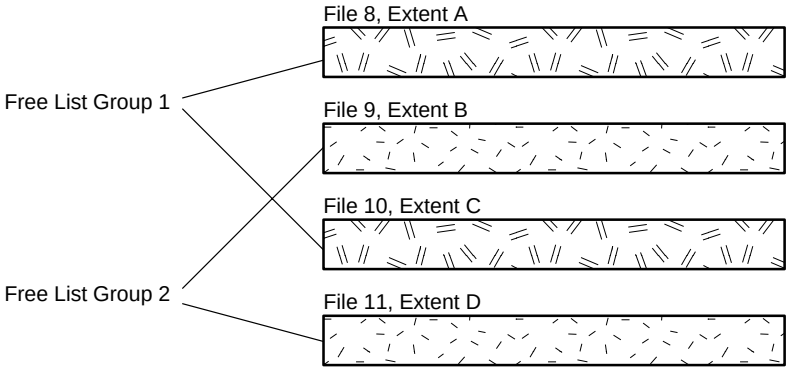
If each extent in the table is in a separate datafile, you can use the GC_FILES_TO_LOCKS parameter to allocate specific ranges of PCM locks to each extent, so that each set of PCM locks is associated with only one group of free lists.

Figure 11–7 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 11–7 Extents and Free List Groups

GC_FILES_TO_LOCKS = 8, 10:10; 9, 11:10



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.

> **See Also:** "Setting GC_FILES_TO_LOCKS: PCM Locks for Each Datafile" on page 15-6.

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.

See Also: "Pre-allocating Extents (Optional)" on page 17-11.

Dynamic Allocation of Blocks on Lock Boundaries

If you primarily need to accommodate growth, the strategy of dynamically allocating blocks to free list groups would be more effective than pre-allocation 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.

Remember that 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.

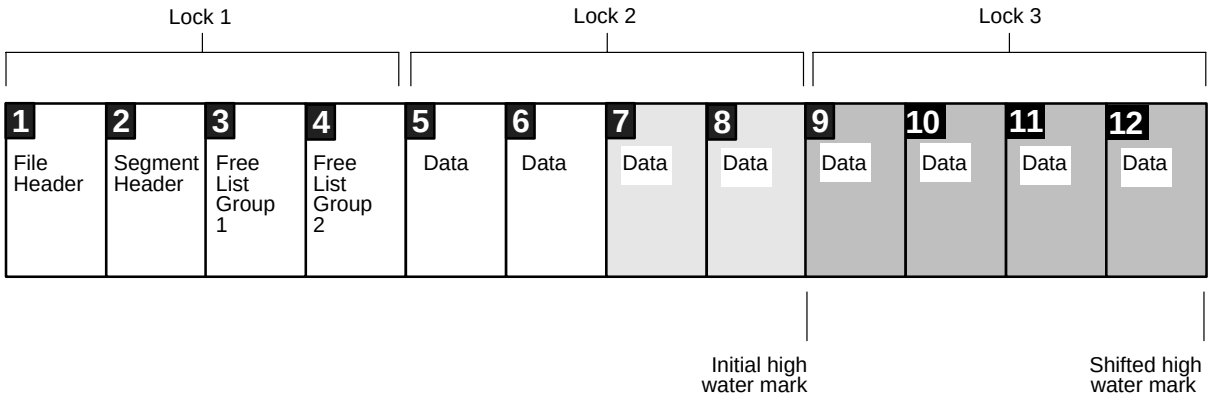
See Also: "Dynamically Allocating Extents" on page 17-15.

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 in which there are 4 blocks per lock (!4). Locks have been allocated before the block content has been entered. If we have filled datablock D2, held by Lock 2, and then allocate another range of 4 blocks, only the number of blocks fitting within the lock boundary are actually allocated. In this case, this includes blocks 7 and 8. Both of these are protected by your 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 11–8 A File with High Water Mark Moving as Blocks Are Allocated



Example The example in this section assumes that GC_FILES_TO_LOCKS has the following setting for both instances:

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

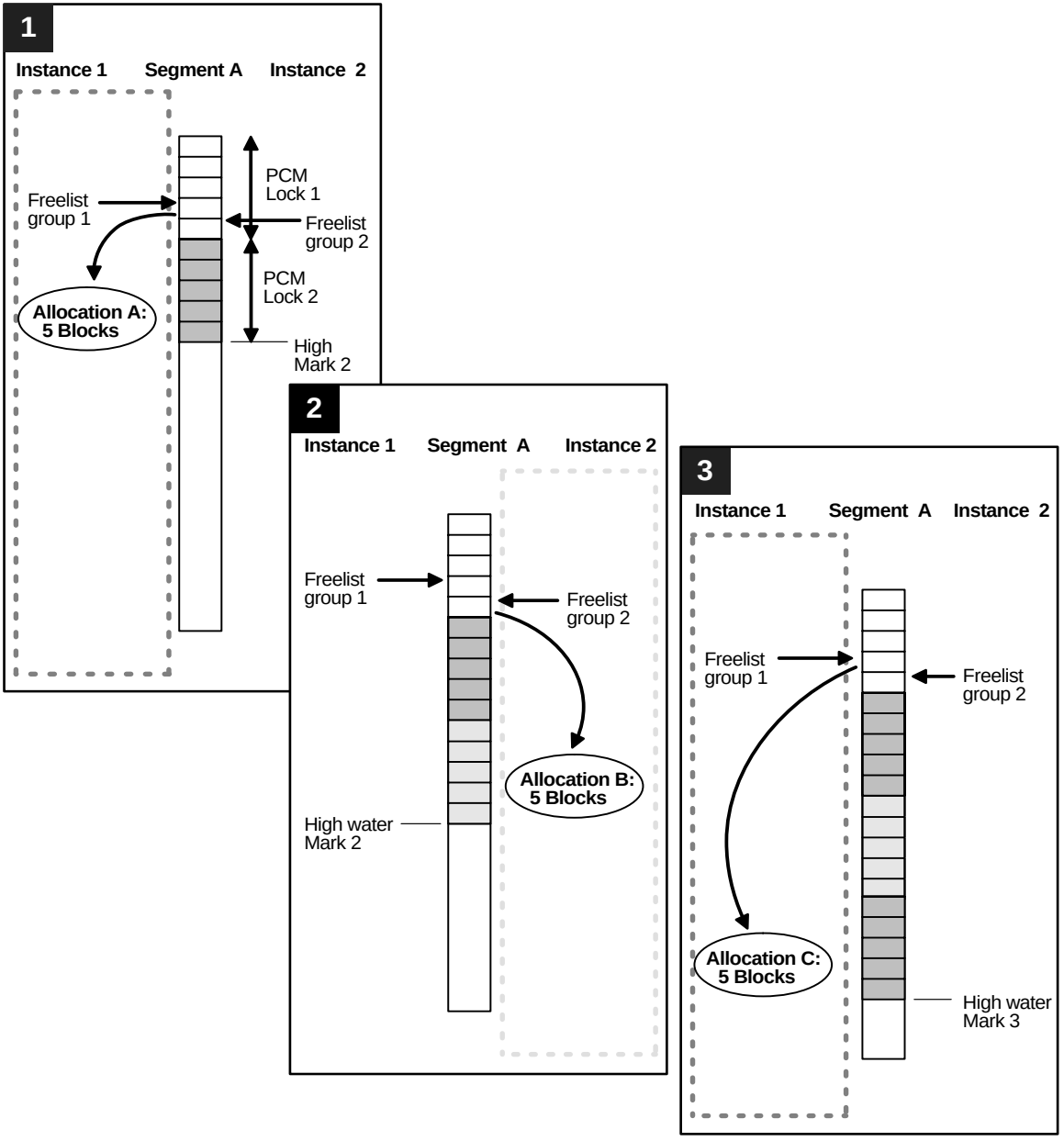
With the EACH option specified, each file in *file_list* is allocated *#locks* number of PCM locks. Within each file, !*blocks* specifies the number of contiguous data blocks to be covered by each lock.

Figure 11–9 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 11–9 Allocating Blocks within an Extent



12

Application Analysis

This chapter provides a conceptual framework for optimizing Oracle Parallel Server (OPS) application design. It includes the following sections:

- How Detailed Must Your Analysis Be?
- Understanding Your Application Profile
- Partitioning Guidelines

> **See Also:** *Oracle8i Tuning* for a discussion of performance tuning principles and tuning methods.

How Detailed Must Your Analysis Be?

The level of detail to which you must analyze an application depends on your goals for the use of OPS. If you need OPS to improve overall database throughput, then a detailed analysis of the database design and application workload profile will be necessary. This ensures that the additional CPU power provided by each node of OPS is fully used for application processing. Even if you are using OPS primarily to provide high availability, careful analysis will enable you to predict the required resources.

Experience gained over many benchmarks and real applications shows that for optimal performance, OPS systems must minimize the computing resources used for parallel cache management. This means minimizing the number of instance lock operations. A successful OPS implementation ensures each node performs very few instance lock operations and subsequently the machine-to-machine high speed interconnect traffic is within the design limitations of the cluster.

You cannot successfully minimize the number of PCM lock operations during the final fine tuning phase of the database lifetime. Rather, you must plan this early in the physical database design process.

> **See Also:** Chapter 13, "Designing Databases for Parallel Server", for a case study showing how to design applications to take advantage of OPS.

Understanding Your Application Profile

To understand your application profile you must classify tables according to application functions and access patterns. This section describes:

- Analyzing Application Functions and Table Access Patterns
- Read-only Tables
- Random SELECT and UPDATE Tables
- INSERT, UPDATE, or DELETE Tables
- Planning the Implementation

The following comments apply equally to clustered tables or non-clustered tables.

Analyzing Application Functions and Table Access Patterns

Beyond performing the usual application and data analysis phases, a database designer for OPS must anticipate the types of transactions or business functions that may cause excessive lock conversion rates. You must cross reference the core application tables and their access patterns with the application functions.

> **See Also:** Chapter 13, "Designing Databases for Parallel Server", for worksheets you can use to analyze table access patterns.

Read-only Tables

With tables that are predominantly read-only, all OPS nodes quickly initialize the PCM locks to shared mode and very little lock activity takes place. Read-only tables and their associated index structures require the allocation of very few PCM locks. With this table type you can expect good performance and scalability with OPS.

Also consider putting tables in read-only tablespaces, using the SQL statement ALTER TABLESPACE READ ONLY. This has several advantages: it speeds up access of the particular tablespace and overall recovery, PCM locks are not required, and you only need to back up a tablespace once after you make it made read-only.

Scalability of parallel query in OPS is subject to the interconnect speed between the nodes. You may need to run high levels of parallelism just to keep the processors

busy. It is not unusual to run a degree of parallelism three times the number of nodes (or processors).

These files should have their own PCM lock as specified in the GC_FILES_TO_LOCKS parameter, even if the application is read-only. Large sorts, such as queries using SORT MERGE JOINs, or sorts with GROUP-BYs and ORDER-BYs, can update the data dictionary in the SYSTEM tablespace.

> **See Also:** "The Four Levels of Scalability" on 2 - 1 and "Setting the Degree of Parallelism" in *Oracle8i Tuning*.

Random SELECT and UPDATE Tables

Random SELECT and UPDATE tables, or non-partitioned tables, have transactions that may read and then update any of the rows in a table. This type of access requires many lock conversions. First, the instance executing the transaction must obtain a shared PCM lock on the data block. This lock request may cause a lock downgrade operation on another node. The instance executing the transaction must finally obtain an exclusive mode PCM lock when the UPDATE is actually performed.

If user transactions on different nodes modify data blocks locked by the same PCM lock concurrently, there will be a noticeable performance penalty. In some cases you can reduce contention by creating additional hashed PCM locks. In large tables, however, hardware and practical limitations may mean that the number of hashed PCM locks you can effectively use may be limited. For example, to reduce false contention you would need millions of hashed PCM locks. However, memory limitations and startup time make this impossible. In these cases, fine grained or releasable hashed locks may be a good alternative.

For this type of table, if none of the table's index keys are updated, then the index's PCM locks are only converted to shared mode and thus require few PCM locks.

> **See Also:** For more information, please refer to "Implement Hashed or Fine Grain Locking" on page 13-17.

INSERT, UPDATE, or DELETE Tables

Transactions on random INSERT, UPDATE and DELETE tables require reading a number of data blocks and then modifying some or all of the data blocks read. This process for each of the data blocks specified again requires converting the PCM lock to shared mode and then converting it to exclusive mode upon block modification. This process has the same performance issues as random SELECT and UPDATE tables.

For this type of table more performance issues exist for two main reasons: index data blocks are changed, and contention occurs for data blocks on the table's free list.

In INSERT, DELETE and UPDATE transactions that modify indexed keys, you need to maintain the table's indexes. This process requires the modification of additional index blocks--and so the number of potential lock converts increases. In addition, index blocks probably require additional lock converts since users on other nodes will be using the index to access other data. This applies particularly to the initial root components of the index where block splitting may be taking place. This causes more lock converts from null to exclusive and vice versa on all nodes within the cluster.

If the INSERT and DELETE operations are subject to long-running transactions, then there is a high chance that another instance will require read consistency information to complete its transactions. This process forces yet more lock conversions as rollback segment data blocks are flushed to disk and are made available to other instances.

Index block contention involving high lock convert rates must be avoided at all costs if performance is a critical issue in your OPS implementation.

Index block contention can be made more extreme when using a sequence number generator to generate unique keys for a table from multiple OPS nodes. When generating unique keys, make the instance number part of the primary key so each instance performs INSERTs into a different part of the index. Spreading the INSERT load over the full width of the index can improve both single and multiple instance performance. Do this using reverse key indexes.

In INSERT operations, allocation of free space within an extent may also cause high lock convert rates. This is because multiple instances may wish to insert new rows into the same data blocks, or into data blocks that are close together. Contention occurs if these data blocks are managed by the same PCM lock. To avoid this, either partition the tables and indexes so different instances use them, or create tables to allow use of multiple free lists and multiple free list groups.

See Also: Chapter 17, "Using Free List Groups to Partition Data" and *Oracle8i Concepts.*

Planning the Implementation

Having analyzed the application workload, you can now plan the application's OPS implementation. Using the access profile you can see which transactions will run well over multiple nodes, and which transactions should be executed within a single node. In many cases compromises and trade-offs are required to ensure that the application performs as needed.

Note: Load balancing between nodes should not be the main objective. Whereas load balancing is useful in benchmarking situations, it may not be useful in a real-world application. Partitioning is the key to performance in OPS.

Partitioning Guidelines

This section covers the following topics:

- Overview
- Application Partitioning
- Data Partitioning

Overview

The database designer must clearly understand the system performance implications and design trade-offs made by application partitioning. Always bear in mind that your goal is to minimize synchronization: this will result in optimized performance.

As noted earlier, if you minimize the number of lock conversions, OPS' performance will be predictable and scalable. By partitioning the application and/or data, you can create and maintain cache affinities of database data with respect to specific nodes of a cluster. A partitioned application ensures that a minimum number of lock conversions are performed, thus pings and Integrated Distributed Lock Manager (IDLM) activity should be very minimal. If excessive IDLM lock activity occurs in a partitioned application, your partitioning strategy may be inappropriate, or the database creation and tuning process was incorrect.

Application Partitioning

Many partitioning techniques exist to achieve high system performance. One of the simplest ways to break up or partition the load upon the database is to run different applications that access the same database on different nodes of the cluster. For example, one application may only reference a fixed set of tables that reside in one set of datafiles, and another application may reference a different set of tables residing in a different set of datafiles. These applications can be run on different nodes of a cluster and should yield good performance if the datafiles are assigned different PCM locks. There will be no conflict for the same database objects (since they are in different files) and hence no conflict for the same database blocks.

This scenario is particularly applicable to applications that during the day support many users and high OLTP workloads, and during the night run high batch and decision support workloads. In this case, you can partition applications among the cluster nodes to sustain good OLTP performance during the day.

This model is similar to a distributed database model, where tables that are accessed together are stored together. At night, when it is necessary to access tables that may be partitioned for OLTP purposes, you still can exploit the advantages of a single database: all the data is stored effectively within a single database. This should provide improved batch and decision support, better query performance, reduced network traffic, and fewer data replication issues.

With this approach, you must ensure that each application's tables and indexes are stored such that one PCM lock does not cover any data blocks used by both applications. Should this happen the benefit of partitioning would be lost. To correct the situation, store each application's table and index data in separate datafiles.

Applications sharing a set of SQL statements perform best when they run on the same instance. Because shared SQL areas are not shared across instances, similar sets of SQL statements should run on one instance to improve memory usage and reduce parsing.

Data Partitioning

Sometimes the partitioning of applications between nodes may not be possible. As an alternative approach, you can partition the database objects themselves. To do this effectively, you must analyze the application profile in depth. You may or may not need to split a table into multiple tables. In OPS, the partitioning process can involve horizontal partitioning of the table between predefined key ranges.

In addition to partitioning and splitting database objects, ensure that each user transaction is executed by the correct OPS instance. The correct node for execution of the transaction is a function of the actual data values being used in the transaction. This process is more commonly known as *data-dependent routing*.

The process of partitioning a table for purposes of increasing parallel server performance brings with it various development and administration implications.

From a development perspective, as soon as the table is partitioned, the quantity and complexity of application code increases. In addition, partitioning a table may compromise the performance of other application functions, such as batch and decision support queries.

You can accomplish data-dependent routing in one of two ways: if the partitioning of the tables fits well within actual partition usage patterns, in other words, you partitioned the table by state or call center, and users are similarly partitionable, then you can accomplish manual routing by having users connect to the correct instance. Otherwise, the administration of data-dependent routing may be complex and can involve additional application code.

You can simplify the process if the application uses a transaction processing monitor (TPM) or RPC mechanism. It is possible to code into the configuration of the TPM a data-dependent routing strategy based on the input RPC arguments. Similarly, this process could be coded into procedural code using a case statement to determine which instance should execute the transaction.

> **See Also:** "Client-Server Systems" on page 1-20 and *Oracle8i Tuning* for more information about partitioning.

Part III

Oracle Parallel Server Development Procedures

13

Designing Databases for Parallel Server

This chapter prescribes a general methodology for designing systems optimized for Oracle Parallel Server (OPS).

- Overview
- Case Study: From Initial Database Design to OPS
- Analyze Access to Tables
- Analyze Transaction Volume by Users
- Partition Users and Data
- Partition Indexes
- Implement Hashed or Fine Grain Locking
- Implement and Tune Your Design

Overview

This chapter provides techniques for designing new applications for use with OPS. You can also use these analytical techniques to evaluate existing applications and see how well suited they are for migration to a parallel server.

Note: Always remember that your goal is to minimize contention: doing so results in optimized performance.

This chapter assumes you have made an initial database design. To optimize your design for OPS, follow the methodology suggested here.

1. Develop an initial database design.
2. Analyze access to tables.
3. Analyze transaction volume.
4. Decide how to partition users and data.
5. Decide how to partition indexes, if necessary.
6. Choose hashed or fine grain locking.
7. Implement and tune your design.

Case Study: From Initial Database Design to OPS

A case study is used is this chapter to demonstrate analytical techniques in practice. Although your applications will differ, this example helps you to understand the process.

- "Eddie Bean" Catalog Sales
- Tables
- Users
- Application Profile

"Eddie Bean" Catalog Sales

The case study concerns the "Eddie Bean" catalog sales company, which has many order entry clerks processing telephone orders for various products. Shipping clerks fill orders and accounts receivable clerks handle billing. Accounts payable clerks handle orders for supplies and services the company requires internally. Sales managers and financial analysts run reports on the data. This company's financial application has three business processes operating on a single database:

- Order entry
- Accounts payable
- Accounts receivable

Tables

Tables from the Eddie Bean database include:

Table 13–1 "Eddie Bean" Sample Tables

Table	Contents
ORDER_HEADER	Order number, customer name and address.
ORDER_ITEMS	Products ordered, quantity, and price.
ORGANIZATIONS	Names, addresses, phone numbers of customers and suppliers.
ACCOUNTS_PAYABLE	Tracks the company's internal purchase orders and payments for supplies and services.
BUDGET	Balance sheet of the company's expenses and income.
FORECASTS	Projects future sales and records current performance.

Users

Various application users access the database to perform different functions:

- Order entry clerks
- Accounts payable clerks
- Accounts receivable clerks
- Shipping clerks
- Sales manager
- Financial analyst

Application Profile

Operation of the Eddie Bean application is fairly consistent throughout the day: order entry, order processing, and shipping are performed all day. These functions are not for example, segregated into separate one-hour time slots.

About 500 orders are entered per day. Each order header is updated about 4 times during its lifetime. So we expect about 4 times as many updates as inserts. There are many selects, because many employees are querying order headers: people doing sales work, financial work, shipping, tracing the status of orders, and so on.

There are on average 4 items per order. Order items are never updated: an item may be deleted and another item entered.

The ORDER_HEADER table has four indexes. Each of the other tables has a primary key index only.

Budget and Forecast activity has a much lower volume than the order tables. They are read frequently, but modified infrequently. Forecasts are updated more often than Budget, and are deleted once they go into actuals.

The vast bulk of the deletes are performed as a nightly batch job. This maintenance activity does not, therefore, need to be included in the analysis of normal functioning of the application.

Analyze Access to Tables

Begin by analyzing the existing (or expected) access patterns for tables in your database. Then decide how to partition the tables and group them according to access pattern.

- Table Access Analysis Worksheet
- Case Study: Table Access Analysis

Table Access Analysis Worksheet

List all your high-activity database tables in a worksheet like the one shown in Table 13–2:

Table 13–2 Table Access Analysis Worksheet

	Daily Access Volume							
	Read Access		Write Access					
	Select		Insert		Update		Delete	
Table Name	Operations	I/Os	Operations	I/Os	Operations	I/Os	Operations	I/Os

To complete this worksheet, estimate the volume of each type of operations. Then calculate the number of reads and writes (I/Os) the operations entail.

Estimating Volume of Operations

For each type of operation to be performed on a table, enter a value reflecting *the normal volume you would expect in the course of a day.*

> **Note:** The emphasis throughout this analysis is on *relative values*—gross figures describing the normal use of an application. Even if an application does not yet exist, you can project the types of users and estimate relative levels of activity. Maintenance activity on the tables is not generally relevant to this analysis.

Calculating I/Os per Operation

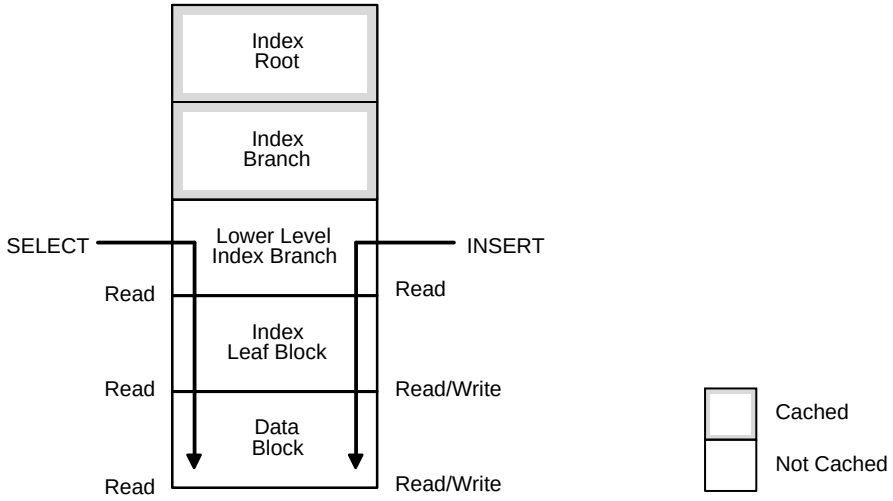
For each value in the Operations column, calculate the number of I/Os that will be generated using a worst-case scenario.

The SELECT operation involves read access, and the INSERT, UPDATE and DELETE operations involve both read and write access. These operations access not only data blocks, but also any related index blocks.

> **Note:** The number of I/Os generated per operation changes *by table* depending on the access path of the table, and the table's size. It also changes depending on the number of indexes a table has. A small index, for example, may have only a single index branch block.

For example, Figure 13–1 illustrates read and write access to data in a large table in which two levels of the index are not in the buffer cache and only a high level index is cached in the SGA.

Figure 13–1 Number of I/So per SELECT or INSERT Operation



In this example, assuming that you are accessing data by way of the primary key, a SELECT entails three I/Os:

1. One I/O to read the first lower level index block.
2. One I/O to read the second lower level index block.
3. One I/O to read the data block.

> **Note:** If all of the root and branch blocks are in the SGA, a SELECT may entail only two I/Os: read leaf index block, read data block.

An INSERT or DELETE statement entails at least five I/Os:

1. One I/O to read the data block.
2. One I/O to write the data block.
3. Three I/Os *per index*: 2 to read the index entries and 1 to write the index.

One UPDATE in this example entails seven I/Os:

1. One I/O to read the first lower level index block.
2. One I/O to read the second lower level index block.

3. One I/O to read the data block.
4. One I/O to write the data block.
5. One I/O to read the first lower level index block again.
6. One I/O to read the second lower level index block again.
7. One I/O to write the index block.

> **Note:** An INSERT or DELETE affects *all* indexes, but an UPDATE sometimes may affect only *one* index. Check the number of changed index keys.

I/Os per Operation for Sample Tables

In the case study, the number of I/Os per operation differs from table to table because the number of indexes differs from table to table.

Table 13–3 shows how many I/Os are generated by each type of operation on the ORDER_HEADER table. It assumes that the ORDER_HEADER table has four indexes.

Table 13–3 Number of I/Os per Operation: Sample ORDER_HEADER Table

Operation	SELECT	INSERT	UPDATE	DELETE
Type of Access	read	read/write	read/write	read/write
Number of I/Os	3	14	7	14

> **Note:** You must adjust these figures depending upon the actual number of indexes and access path for each table in your database.

Table 13–4 shows how many I/Os generated per operation for each of the other tables in the case study, assuming each of them has a primary key index only.

Table 13–4 Number of I/Os per Operation: Other Sample Tables

Operation	SELECT	INSERT	UPDATE	DELETE
Type of Access	read	read/write	read/write	read/write
Number of I/Os	3	5	7	5

For the purposes of this analysis, you can disregard the fact that changes made to data also generate rollback segments, entailing additional I/Os. These I/Os are instance-based. Therefore, they should not cause problems with your OPS application.

See Also: *Oracle8i Concepts* for more information about indexes.

Case Study: Table Access Analysis

Table 13–5 shows rough figures reflecting normal use of the application in the case study.

Table 13–5 Case Study: Table Access Analysis Worksheet

	Daily Access Volume							
	Read Access		Write Access					
	Select		Insert		Update		Delete	
Table Name	Operations	I/Os	Operations	I/Os	Operations	I/Os	Operations	I/Os
ORDER_HEADER	20,000	60,000	500	7,000	2,000	14,000	1,000	14,000
ORDER_ITEM	60,000	180,000	2,000	10,000	0	0	4,030	20,150
ORGANIZATIONS	40,000	120,000	10	50	100	700	0	0
BUDGET	300	900	1	5	2	14	0	0
FORECASTS	500	1,500	1	5	10	70	2	10
ACCOUNTS_PAYABLE	230	690	50	250	20	140	0	0

The following conclusions can be drawn from this table:

- Only the ORDER_HEADER and ORDER_ITEM tables have significant levels of write access.
- ORGANIZATIONS, by contrast, is predominantly a read-only table. While a certain number of INSERT, UPDATE, and DELETE operations will maintain it, its normal use is SELECT-only.

Analyze Transaction Volume by Users

Begin by analyzing the existing (or expected) access patterns for tables in your database. Then decide how to partition the tables and group them according to access pattern.

- Transaction Volume Analysis Worksheet
- Case Study: Transaction Volume Analysis

Transaction Volume Analysis Worksheet

For each table with a high volume of write access, analyze the transaction volume per day for each type of user.

Note: For read-only tables, you do *not* need to analyze transaction volume by user type.

Use worksheets like the one in Table 13–6:

Table 13–6 Transaction Volume Analysis Worksheet

Table Name:									
		Daily Transaction Volume							
		Read Access		**Write Access**					
		Select		**Insert**		**Update**		**Delete**	
Type of User	**No.Users**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**

Begin by estimating the volume of transactions by each type of user and then calculate the number of I/Os entailed.

Case Study: Transaction Volume Analysis

The following tables show transaction volume analysis of the three tables in the case study that have a high level of write access: ORDER_HEADER, ORDER_ITEMS, and ACCOUNTS_PAYABLE.

ORDER_HEADER Table

Table 13–7 shows rough estimates for values in the ORDER_HEADER table in the case study.

Table 13–7 Case Study: Transaction Volume Analysis: ORDER_HEADER Table

Table Name: ORDER_HEADER									
		Daily Transaction Volume							
		Read Access		**Write Access**					
Type of	**No.**	**Select**		**Insert**		**Update**		**Delete**	
User	**Users**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	5,000	15,000	500	7,000	0	0	0	0
Accounts payable clerk	5	0	0	0	0	0	0	0	0
Accounts receivable clerk	5	6,000	18,000	0	0	1,000	7,000	0	0
Shipping clerk	4	4,000	12,000	0	0	1,000	7,000	0	0
Sales manager	2	3,000	9,000	0	0	0	0	0	0
Financial analyst	2	2,000	6,000	0	0	0	0	0	0

The following conclusions can be drawn from this table:

- Order entry clerks perform all inserts on this table.
- Accounts receivable and shipping clerks perform all updates.
- Sales managers and financial analysts only perform select operations.
- Accounts payable clerks never use the table.

Deletes are performed as a maintenance operation, so you do not need to consider them in this analysis. Furthermore, the application developers realize that sales managers normally access data for the current month, whereas financial analysts access mostly historical data.

ORDER_ITEMS Table

Table 13–8 shows rough estimates for values in the ORDER_ITEMS table in the case study.

Table 13–8 Case Study: Transaction Volume Analysis: ORDER_ITEMS Table

Table Name: ORDER_ITEMS									
Type of User	**No. Users**	**Daily Transaction Volume**							
		Read Access		**Write Access**					
		Select		**Insert**		**Update**		**Delete**	
		Operations	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	15,000	45,000	2,000	10,000	0	0	20	100
Accounts payable clerk	5	0	0	0	0	0	0	0	0
Accounts receivable clerk	5	18,000	54,000	0	0	0	0	10	50
Shipping clerk	4	12,000	36,000	0	0	0	0	0	0
Sales manager	2	9,000	27,000	0	0	0	0	0	0
Financial analyst	2	6,000	18,000	0	0	0	0	0	0

The following conclusions can be drawn from this table:

- Order entry clerks perform all inserts on this table.
- Updates are rarely performed
- Accounts receivable clerks, shipping clerks, sales managers and financial analysts perform a heavy volume of select operations on the table.
- Accounts payable clerks never use the table.

The ORDER_HEADER table has more writes than ORDER_ITEMS because the order header tends to require more changes of status, such as address changes, than the list of available products. The ORDER_ITEM table is seldom updated because new items are listed as journal entries.

ACCOUNTS_PAYABLE Table

Table 13–9 shows rough figures for the ACCOUNTS_PAYABLE table in the case study. Although this table does not have a particularly high level of write access, we have analyzed it because it contains the main operation that the accounts payable clerks perform.

Table 13–9 Case Study: Transaction Volume Analysis: ACCOUNTS_PAYABLE Table

Table Name: ACCOUNTS_PAYABLE									
		Daily Transaction Volume							
		Read Access		**Write Access**					
Type of	**No.**	**Select**		**Insert**		**Update**		**Delete**	
User	**Users**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	0	0	0	0	0	0	0	0
Accounts payable clerk	5	200	600	50	250	20	140	0	0
Accounts receivable clerk	5	0	0	0	0	0	0	0	0
Shipping clerk	4	0	0	0	0	0	0	0	0
Sales manager	2	0	0	0	0	0	0	0	0
Financial analyst	2	30	90	0	0	0	0	0	0

The following conclusions can be drawn from this table:

- Accounts payable clerks send about 50 purchase orders per day to suppliers. These clerks are the only users who change the data in this table.
- Financial analysts occasionally study the information.

Deletes are performed as a maintenance operation, so you do not need to consider them in this analysis.

Partition Users and Data

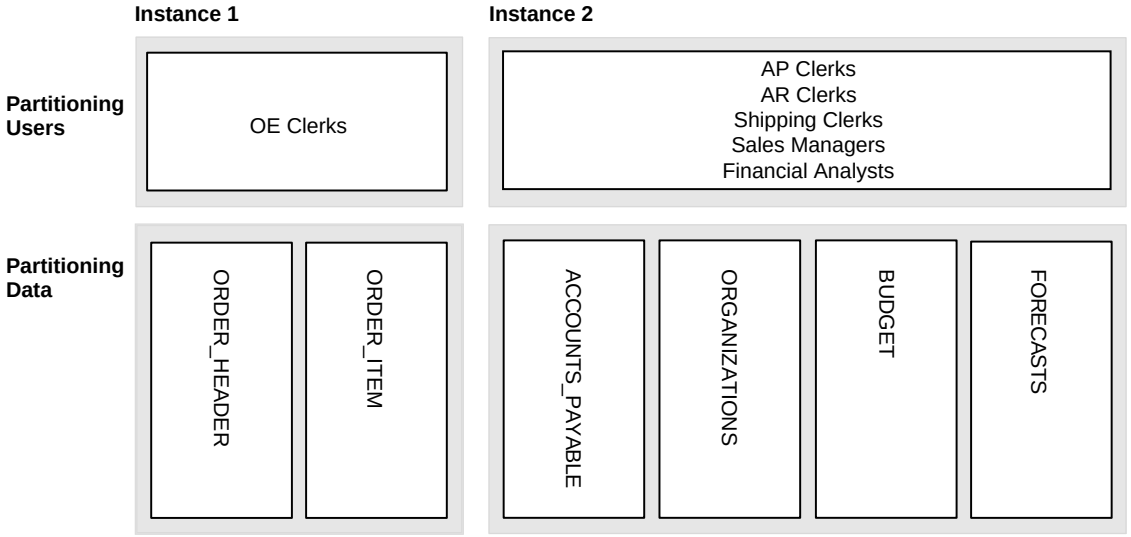
Your goal is to partition applications across instances. This can involve separating types of users across instances and partitioning data that needs to be written only by certain types of users. This minimizes the amount of contention on your system. This section covers:

- Case Study: Initial Partitioning Plan
- Case Study: Further Partitioning Plans

Case Study: Initial Partitioning Plan

In the case study, for example, the large number of order entry clerks doing heavy insert activity on the ORDER_HEADER and ORDER_ITEM tables should not be separated across machines. You should concentrate these users on one node along with the two tables they use most. A good starting point, then, would be to set aside one node for the OE clerks, and one node for all other users as illustrated in Figure 13–2.

Figure 13–2 Case Study: Partitioning Users and Data



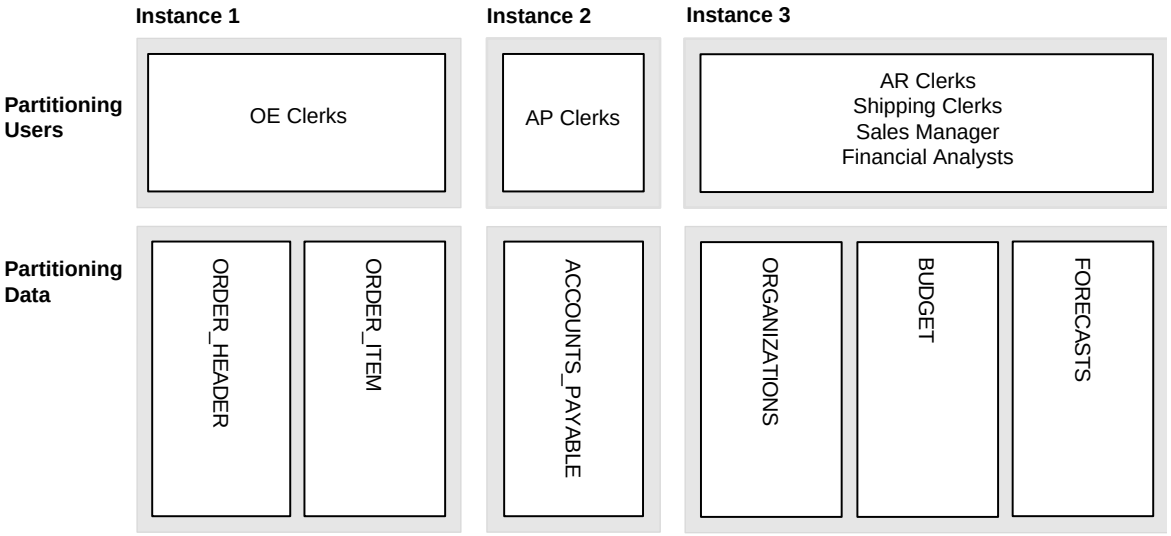
This system is probably well balanced across nodes. The database intensive reporting done by financial analysts takes a good deal of system resources, whereas the transactions run by the order entry clerks are relatively simple.

The load balancing by manipulating the number of users across the system is typically useful, but not always critical. Load balancing has a lower priority for tuning than reducing contention.

Case Study: Further Partitioning Plans

In the case study it is also clear that accounts payable data is written exclusively by accounts payable clerks. You can thus effectively partition this data the set of users onto a separate instance as shown in Figure 13–3.

Figure 13–3 Case Study: Partitioning Users and Data: Design Option 1



When all users needing write access to a certain part of the data are concentrated on one node, the PCM locks all reside on that node. In this way, lock ownership is not switching back and forth between instances.

Based on this analysis, you primarily have two design options.

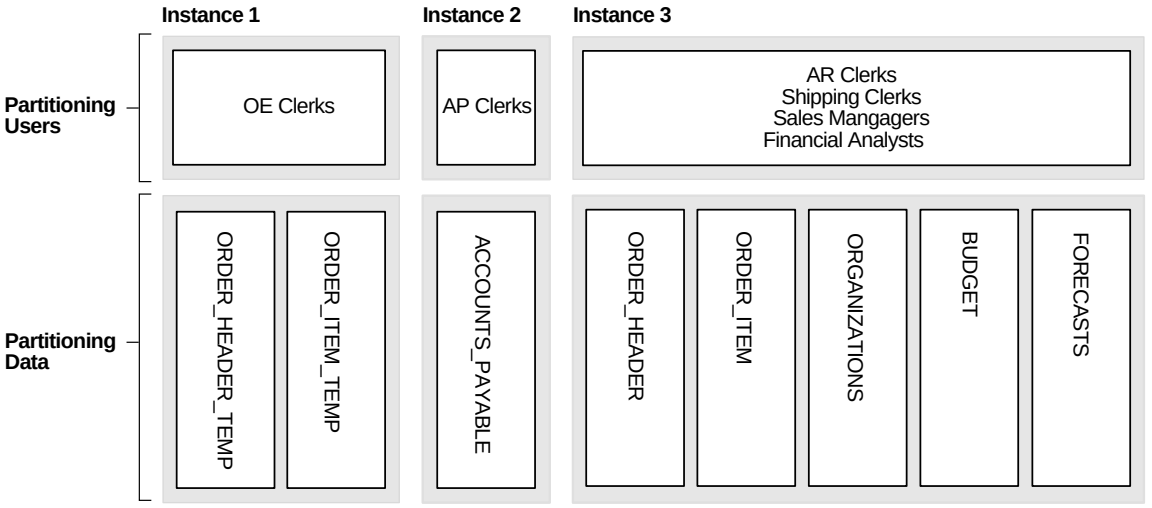
Design Option 1

You can set up your as shown above with all order entry clerks on one instance to minimize contention for exclusive PCM locks on the table. This allows sales managers and financial analysts to get up-to-the-minute information. Since they do want data that is predominantly historical, there should not be too much contention for current records.

Design Option 2

Alternatively, you could implement a separate temporary table for ORDER_ITEM / ORDER_HEADER. This table is only for recording new order information. Overnight, you could incorporate changes into the main table against which all queries are performed. This solution would work well if it is not vitally important that financial analysis have current data. This is probably true only if they are primarily interested in looking at historical data. This would not be appropriate if the financial analysts needed up-to-the-minute data.

Figure 13–4 Case Study: Partitioning Users and Data: Design Option 2



Partition Indexes

You need to consider index partitioning if multiple nodes in your system are inserting into the same index. In this situation, you must ensure that different instances insert into different points within the index.

Note: This problem is avoided in the Eddie Bean case study because application and data usage are partitioned.

See Also: "Creating Free Lists for Indexes" on page 17-8 for tips on using free lists, free list groups, and sequence numbers to avoid contention on indexes. For more information about indexes as a point of contention, please see "Locating Lock Contention within Applications" on page 19-4. Also refer to *Oracle8i Concepts* for tips on how to physically partition a table and an instance to avoid the use of free list groups.

Implement Hashed or Fine Grain Locking

For many applications, the DBA needs to decide whether to use hashed or fine grain locking for particular database files.

You should design for the worst case scenario that would use hashed locking. Then, in the design or monitoring phases, if you discover a situation where you have too many locks, or if you suspect false pings, you should try fine grain locking.

Begin with an analysis at the database level. You can use a worksheet like the one shown in Table 13–10:

Table 13–10 Worksheet: Database Analysis for Hashed or Fine Grain Locking

Block Class	Relevant Parameter(s)	Use Fine Grain or Hashed Locking?

Next, list the files and database objects in a worksheet like the one shown in Table 13–11. Decide which locking mode to use for each file.

Table 13–11 Worksheet: When to Use Hashed or Fine Grain Locking

Filename	Objects Contained	Use Fine Grain or Hashed Locking?

See Also: "Applying Fine Grain and Hashed Locking to Different Files" on page 9-21.

Implement and Tune Your Design

Up to this point, you conducted an analysis using estimated figures. To finalize your design you must now either prototype the application or actually implement it and get it running. By observing the actual system, you can tune it further.

To do this, try the following techniques:

- Identify blocks that are being pinged and determine where contention exists.
- Consider moving users from one instance to another to reduce the amount of pinging and false pinging.
- If you detect a high level of false pinging, consider increasing the granularity of the locks by placing more locks on each file.
- If there is pinging on inserts, adjust the free lists or use multiple sequence number generators so that inserts occur in different parts of the index.

> **See Also:** Chapter 19, "Tuning to Optimize Performance" in *Oracle8i Tuning*.

14

Creating a Database and Objects for Multiple Instances

This chapter describes:

- Creating a Database for a Multi-instance Environment
- Creating Database Objects to Support Multiple Instances
- Changing the Value of CREATE DATABASE Options

Creating a Database for a Multi-instance Environment

This section covers aspects of database creation specific to OPS:

- Summary of Tasks
- Setting Initialization Parameters for Database Creation
- Database Creation and Start Up
- Setting CREATE DATABASE Options

Summary of Tasks

Database creation tasks specific to OPS can be summarized as follows:

1. Set initialization parameters for every instance, including log archiving.
2. With parallel server disabled, enter the CREATE DATABASE statement, setting MAXINSTANCES and other important options specific to OPS.
3. Create rollback segments for each node.

4. Dismount the database then remount it with the parameter setting PARALLEL_SERVER=TRUE in the initialization file. Then start up the parallel server.

   See Also: "Creating a Database" in the *Oracle8i Administrator's Guide.*

Setting Initialization Parameters for Database Creation

Certain initialization parameters critical at database creation or that affect certain database operations must have the same value for every instance in OPS. Be sure these are identical across all instances before creating a database for a multi-instance environment.

Using ARCHIVELOG Mode

To enable the ARCH process while creating a database, set the initialization parameter LOG_ARCHIVE_START to TRUE. Then change the mode to ARCHIVELOG with the ALTER DATABASE statement before starting up the instance that creates the database.

Alternatively, you can reduce overhead by creating the database in NOARCHIVELOG mode. This is the default. Then change to ARHIVELOG mode.

You cannot use the STARTUP command to change the database archiving mode. After creating a database, use the following Server Manager commands to change to archiving mode and reopen the database with parallel server enabled:

```
ALTER DATABASE CLOSE;
ALTER DATABASE ARCHIVELOG;
SHUTDOWN;
STARTUP;
```

See Also: "Archiving the Redo Log Files" on page 21-2 and "Parameters that Must Be Identical on All Instances" on page 18-11.

Database Creation and Start Up

Use the standard procedure to create a database.

> **Note:** The CREATE DATABASE statement mounts and opens the newly created database, leaving the parallel server disabled. You must close and dismount the database, then remount it with parallel server enabled.

1. Start Server Manager.
2. Connect with SYSDBA privileges.
3. Start up an instance with the NOMOUNT option.
4. Issue the CREATE DATABASE statement.
5. Create additional rollback segments and threads, as needed.
6. Close and dismount the database.

   ```
   SHUTDOWN
   ```

7. Update the initialization files to be sure they point to the proper rollback segments and threads,
8. Make sure parallel server is enabled.
9. Remount the database.

   ```
   STARTUP [ OPEN databasename ]
   ```

See Also: "Starting Instances" on page 18-13.

Setting CREATE DATABASE Options

This section describes CREATE DATABASE options specific to OPS.

Setting MAXINSTANCES

The MAXINSTANCES option of CREATE DATABASE limits the number of instances that can access a database concurrently. MAXINSTANCES defaults to the maximum value specific to your operating system; on most systems the default is two.

For OPS, set MAXINSTANCES to a value greater than the maximum number of instances you expect to run concurrently. This way, if instance A fails and is being recovered by instance B, you will be able to start instance C before instance A is fully recovered.

Setting MAXLOGFILES and MAXLOGMEMBERS

The MAXLOGFILES option of CREATE DATABASE specifies the maximum number of redo log groups that can be created for the database, and the MAXLOGMEMBERS option specifies the maximum number of members or copies per group.

For OPS, set MAXLOGFILES to the maximum number of threads possible, multiplied by the maximum anticipated number of groups per thread.

Setting MAXLOGHISTORY

The MAXLOGHISTORY option of CREATE DATABASE specifies the maximum number of redo log files that can be recorded in the log history of the control file. The log history is used for automatic media recovery of OPS.

For OPS, you should set MAXLOGHISTORY to a large value, such as 1000. The control files can then only store information about this number of redo log files. When the log history exceeds this limit, old history entries are overwritten. The default for MAXLOGHISTORY is zero, which disables the log history.

Setting MAXDATAFILES

The MAXDATAFILES option is generic, but OPS tends to have more data files and log files than standard systems. On your platform the default value of this option may be too low.

> **See Also:** *Oracle8i SQL Reference* for complete descriptions of the SQL statements CREATE DATABASE and ALTER DATABASE. Also see your Oracle operating system-specific documentation for information on default values of CREATE DATABASE options. For more information about redo log groups and members, see "Redo Log Files" on page 6-3. Also see "Redo Log History in the Control File" on page 21-6 for more information on MAXLOGHISTORY.

Creating Database Objects to Support Multiple Instances

To prepare a new database for OPS, create and configure the following additional database objects.

- Creating Additional Rollback Segments
- Configuring the Online Redo Log for OPS
- Providing Locks for Added Datafiles

Creating Additional Rollback Segments

You must create at least one rollback segment for each instance of a parallel server. To avoid contention, create these rollback segments in a tablespace other than the SYSTEM tablespace.

You must create and bring online one additional rollback segment in the SYSTEM tablespace before creating rollback segments in other tablespaces. The instance that creates the database can create this additional rollback segment and new tablespaces, but it cannot create database objects in non-SYSTEM tablespaces until you bring the additional rollback segment online.

Using Private Rollback Segments

To allocate a private rollback segment to one instance, follow these steps:

1. Create the rollback segment with the SQL statement CREATE ROLLBACK SEGMENT, omitting the keyword PUBLIC. Optionally, before creating the rollback segment you can create a tablespace for it.
2. Specify the rollback segment in the instance's parameter file by naming it as a value for the parameter. This reserves the rollback segment for that instance.
3. Use ALTER ROLLBACK SEGMENT to bring the rollback segment online. You can also restart the instance to use the reserved rollback segment.

A private rollback segment should be specified in only one parameter file so that it is associated with only one instance. If an instance attempts to acquire a private rollback segment that another instance has already acquired, Oracle generates a message and prevents the instance from starting up.

Using Public Rollback Segments

Any instance can create a public rollback segment that can then be *claimed* by any instance when it starts up. Once a rollback segment has been claimed, it is only used

by the instance that claimed it until the instance shuts down, releasing the rollback segment for use by another instance.

To create a public rollback segment, use the SQL statement CREATE PUBLIC ROLLBACK SEGMENT.

Typically, the parameter file for any particular instance does not specify public rollback segments because they are assumed to be available to any instance needing them. However, if another instance is not already using it, you can name a public rollback segment as a value of the ROLLBACK_SEGMENTS parameter.

Public rollback segments are identified as having the owner PUBLIC in the data dictionary view DBA_ROLLBACK_SEGS.

If the parameter file omits the ROLLBACK_SEGMENTS initialization parameter, the instance uses public rollback segments by default.

A public rollback segment is brought online when an instance requiring public rollback segments starts up and acquires it. However, starting an instance using public rollback segments does not ensure that any particular public rollback segment comes online unless the instance acquires all available public rollback segments. Once acquired, a public rollback segment is used exclusively by the acquiring instance.

Bringing online, taking offline, creating, and dropping rollback segments, whether private or public, is the same whether OPS is enabled or disabled.

Private rollback segments stay offline until brought online or until the owning instance restarts. A public rollback segment stays offline until brought online for a specific instance or until an instance requiring a public rollback segment starts up and acquires it.

If you need to keep a public rollback segment offline and do not want to drop it and re-create it, you must prevent other instances that require public rollback segments from starting up.

Monitoring Rollback Segments

You can use the Server Manager command MONITOR ROLLBACK to display information about the status of the rollback segments that the current instance uses.

Alternatively, you can query the dynamic performance views V$ROLLNAME and V$ROLLSTAT for information about the current instance's rollback segments.

Use the Server Manager command CONNECT *@instance-path* to change the current instance before using the MONITOR command or querying the V$ views. You must

have Net8 installed to use the CONNECT command for an instance on a remote node.

You can also query the data dictionary views DBA_ROLLBACK_SEGS and DBA_SEGMENTS for information about the current status of all rollback segments in your database.

For example, to list the current rollback segments, you can query DBA_ROLLBACK_SEGS with the following statement:

```
SELECT segment_name, segment_id, owner, status
   FROM dba_rollback_segs
```

This query displays the rollback segment's name, ID number, owner, and whether it is in use, as shown in the following example:

```
SEGMENT_NAME                SEGMENT_ID     OWNER      STATUS
--------------------------- ----------     ------     ------------
SYSTEM                               0     SYS        ONLINE
PUBLIC_RS                            1     PUBLIC     ONLINE
USERS1_RS                            2     SYS        ONLINE
USERS2_RS                            3     SYS        OFFLINE
USERS3_RS                            4     SYS        ONLINE
USERS4_RS                            5     SYS        ONLINE
PUBLIC2_RS                           6     PUBLIC     OFFLINE
```

In this example, rollback segments identified as owned by user SYS are private rollback segments; the rollback segments identified as owned by user PUBLIC are public rollback segments.

The view DBA_ROLLBACK_SEGS also includes information (not shown) about the tablespace containing the rollback segment, the datafile containing the segment header, and the extent sizes. The view DBA_SEGMENTS includes additional information about the number of extents in each rollback segment and the segment size.

> **See Also:** The *Oracle8i Administrator's Guide* for more information about rollback segments, and about connecting to a database. For the format of the connect string in *instance-path* see *Net8 Administrator's Guide* and your Oracle system-specific documentation. For a description of DBA_ROLLBACK_SEGS and DBA_SEGMENTS, and other dynamic performance views please see the *Oracle8i Reference.*

Configuring the Online Redo Log for OPS

Each database instance has its own "thread" of online redo, consisting of its own online redo log groups. When running OPS, two or more instances concurrently access a single database and each instance must have its own thread. This section explains how to configure these online redo threads for multiple instances with OPS.

You must create each thread with at least two redo log files (or multiplexed groups), and you must enable the thread before an instance can use it.

The CREATE DATABASE statement creates thread number 1 as a public thread and enables it automatically. You must use the ALTER DATABASE statement to create and enable subsequent threads.

Creating Threads

Threads can be either public or private. The initialization parameter THREAD assigns a unique thread number to the instance. If THREAD is zero, which is the default, the instance acquires an available public thread.

Each thread must be created with at least two redo log files or multiplexed groups. You must also enable each thread before an instance can use it**.**

The CREATE DATABASE statement creates thread number 1 as a public thread and enables it automatically. Subsequent threads must be created and enabled with the ALTER DATABASE statement. For example, the following statements create thread 2 with two groups of three members each, as shown in Figure 6–1 on page 6-4:

```
ALTER DATABASE ADD LOGFILE THREAD 2
 GROUP 4 (disk1_file4, disk2_file4, disk3_file4) SIZE 1M REUSE
 GROUP 5 (disk1_file5, disk2_file5, disk3_file5) SIZE 1M REUSE;
ALTER DATABASE ENABLE PUBLIC THREAD 2;
```

If you omit the keyword PUBLIC when you enable the thread, it will be a private thread that cannot be acquired by default. Only one thread number may be specified in the ALTER DATABASE ADD LOGFILE statement, and the THREAD clause must be specified if the thread number of the current instance was chosen by default.

Disabling Threads

You can disable a public or private thread with the statement ALTER DATABASE DISABLE THREAD. You cannot disable a thread if an instance using the thread has the database mounted. To change a thread from public to private, or vice versa, you

must disable the thread and then enable it again. An instance cannot disable its own thread. The database must be open when you disable or enable a thread.

When you disable a thread, Oracle marks its current redo log file as needing to be archived. If you want to drop that file, you might need to first archive it manually.

An error or failure while a thread is being enabled can result in a thread that has a current set of log files but is not enabled. These log files cannot be dropped or archived. In this case, you should disable the thread, even though it is already disabled, then enable it.

Setting the Log's Mode

The mode of using the redo log, ARCHIVELOG or NOARCHIVELOG, is set at database creation. Although rarely necessary, the archive mode can be changed by the SQL statement ALTER DATABASE. When archiving is enabled, online redo log files cannot be reused until they are archived. To switch archiving modes, the database must be mounted with OPS disabled, but the database cannot be open.

The redo log mode is associated with the database rather than with individual instances. For most purposes, all instances should use the same archiving method, either automatic or manual, if the redo log is being used in ARCHIVELOG mode.

Changing the Redo Log

You can change the configuration of the redo log, such as adding, dropping, or renaming a log file or log file member, while the database is mounted with OPS either enabled or disabled. The only restrictions are that you cannot drop or rename a log file or log file member currently in use by any thread. Moreover, you cannot drop a log file if that would reduce the number of log groups to less than two for the thread it is in.

Any instance can add or rename redo log files, or members, of any group for any other instance. As long as there are more than two groups for an instance, a redo log group can be dropped from that instance by any other instance. Changes to redo log files and log members take effect on the next log switch.

See Also: "Archiving the Redo Log Files" on page 21-2.

Providing Locks for Added Datafiles

If datafiles are added while OPS is running, evaluate whether enough locks are available to cover the new files.

Added datafiles use the unassigned locks created when the value for GC_FILES_TO_LOCKS was set. If the remaining locks are not adequate to protect the new files and avoid contention, provide more locks by adjusting the GC* parameters. Performance problems are likely if you neglect to make these adjustments.

In a read-only database, extra locks are not necessary even if you added many new datafiles. In a database heavily used for inserts, however, you might need to provide more locks.

If you determine you need more locks, do the following:

1. Shut down your system.
2. Modify the GC_FILES_TO_LOCKS initialization parameter to provide enough locks for the additional datafiles.
3. Restart the system.

Changing the Value of CREATE DATABASE Options

You can use the CREATE CONTROLFILE statement to change the value of the following database parameters for an existing database:

- MAXINSTANCES
- MAXLOGFILES
- MAXLOGMEMBERS
- MAXLOGHISTORY
- MAXDATAFILES

> **See Also:** *Oracle8i SQL Reference* for a description of the statements CREATE CONTROLFILE and ALTER DATABASE BACKUP CONTROLFILE TO TRACE.

15

Allocating PCM Instance Locks

This chapter explains the init.ora parameters you must set to allocate PCM locks to datafiles for an OPS instance.

- Planning the Use and Maintenance of PCM Locks
- Setting GC_FILES_TO_LOCKS: PCM Locks for Each Datafile
- Tips for Setting GC_FILES_TO_LOCKS
- Setting Other GC_* Parameters
- Tuning PCM Locks

Note: Tuning PCM locks may not be enough to properly scale your application. For more information, please refer to Chapter 12, "Application Analysis"and Chapter 13, "Designing Databases for Parallel Server".

See Also: Chapter 9, "Parallel Cache Management Instance Locks", for a conceptual discussion of PCM locks and GC_* parameters and *Oracle8i Reference* for descriptions of initialization parameters used to allocate locks for OPS.

Planning the Use and Maintenance of PCM Locks

This section describes planning the use and maintenance of PCM locks. It covers:

- Planning and Maintaining Instance Locks
- Key to Allocating PCM Locks
- Examining Datafiles and Data Blocks
- Using Worksheets to Analyze PCM Lock Needs
- Mapping Fixed PCM Locks to Data Blocks
- Partitioning PCM Locks Among Instances

Planning and Maintaining Instance Locks

The IDLM allows you to allocate only a finite number of locks. For this reason you need to analyze and plan for the number of locks your application requires. You also need to know how much memory locks and resources require. Consider these ramifications:

- If you attempt to use more locks than the number configured in the IDLM facility, Oracle will display an error message and shut down the instance.
- If you change Oracle GC_* or LM_* initialization parameters to specify large numbers of locks, this affects the amount of memory used or available in the SGA.
- The number of instances also affects the memory requirements and number of locks needed by your system.

Key to Allocating PCM Locks

The key to assigning locks is to analyze how often data is *changed* using the INSERT, UPDATE, and DELETE commands. You can then determine how to group objects into files based on whether they should be read-only or read / write. Finally, assign locks based on the groupings you have made. In general, follow these guidelines:

- Allocate only a few locks to read-only files.
- Allocate more locks to read / write intensive files.
- If the whole tablespace is read-only, you can simply assign it a single lock. If you did not assign locks to the tablespace, the system would attempt to use spare locks. This can cause contention since the tablespace would contend with

other tablespaces for the spare locks. This can generate unnecessary forced reads / writes.

They key distinction is *not* between types of *objects* (index or table), but between *operations* being performed on an object. The operation dictates the quantity of locks needed.

See Also: Chapter 12, "Application Analysis".

Examining Datafiles and Data Blocks

You must allocate locks at various levels:

- Specify the maximum number of PCM locks to allocate for all datafiles
- Specify how many locks to allocate to blocks in each datafile
- Specify particular locks to cover particular classes of datablocks in a given file

Begin by getting to know your datafiles and the blocks they contain.

How to Determine File ID, Tablespace Name, and Number of Blocks

Use the following command to determine the file ID, file name, tablespace name, and number of blocks for all databases.

```
SELECT FILE_NAME, FILE_ID, TABLESPACE_NAME, BLOCKS
FROM DBA_DATA_FILES;
```

Results are displayed as in the following example:

```
FILE_NAME               FILE_ID     TABLESPACE_NAME     BLOCKS
-------------------------------------------------------------
/v7/data/data01.dbf     1           SYSTEM               200
/v7/data/data02.dbf     2           ROLLBACK            1600
. . .
```

How Many Locks Do You Need?

Use the following approach to estimate the number of locks required for particular uses.

- Consider the nature of the data and the application.

  Many locks are needed on heavily used, concurrently updated datafiles. But a query-only application does not need many locks; a single lock on the datafile suffices.

- Assign many locks to files that many instances modify concurrently.

  This reduces lock contention, minimizes I/O activity, and increases accessibility of the data in the files.

- Assign fewer locks to files that multiple instances do not need to concurrently access.

  This avoids unnecessary lock management overhead.

- Examine the objects in your files, and consider the operations used on them.
- Group read-only objects in read-only tablespace(s).

Using Worksheets to Analyze PCM Lock Needs

On large applications, carefully study the business processes involved. Worksheets similar to those in this section may be useful.

Determine the types of operations your system performs on a daily basis. The distinction between operations needing X locks and those needing S locks is the key. Every time you have to go from one mode to the other, you need locks. Take into consideration the interaction of different instances on a table. Also take into consideration the number of rows in a block, the number of rows in a table, and the growth rate. Based on this analysis, group your objects into files, and assign free list groups.

Figure 15–1 PCM Lock Worksheet 1

Object	Operations needing X mode: Writes			OPS needing S mode: Reads	TS/Datafile
	INSERTS	UPDATES	DELETES	SELECTS	
A		80%		20%	
				Full table scan?	
				Single row?	
B				100%	
C					
D					

Figure 15–2 PCM Lock Worksheet 2

Object	Instance 1	Instance 2	Instance 3
D	INSERT	SELECT	
	UPDATE		
	DELETE		
E			
F			

Figure 15–3 PCM Lock Worksheet 3

Table Name	TS to put it in	Row Size	Number of Columns

Mapping Fixed PCM Locks to Data Blocks

In many cases, you need relatively few PCM locks to cover read-only data compared to data that is updated frequently. This is because read-only data can be shared by all instances of a parallel server. Data that is never updated can be covered by a single PCM lock. Data that is not read-only should be covered by more than a single PCM lock.

If data is read-only, then once an instance owns the PCM locks for the read-only tablespace, the instance never disowns them.The Integrated Distributed Lock Manager (IDLM) operations are not required after the initial lock acquisition.

For best results, partition your read-only tablespace so it is covered by its own set of PCM locks. Do this by placing read-only data in a tablespace that does not have writable data. Then allocate PCM locks to the datafiles in the tablespace using the GC_FILES_TO_LOCKS parameter.

Note: Do not put read-only data and writable data in the same tablespace.

Partitioning PCM Locks Among Instances

You can map PCM locks to particular data blocks to partition PCM locks among instances based on the data each instance accesses.

This technique minimizes unnecessary distributed lock management. Likewise, it minimizes the disk I/O caused by an instance having to write out data blocks because a requested data block was covered by a PCM lock owned by another instance.

For example, if Instance X primarily updates data in datafiles 1, 2, and 3, while Instance Y primarily updates data in datafiles 4 and 5, you can assign one set of PCM locks to files 1, 2, and 3 and another set to files 4 and 5. Then each instance acquires ownership of the PCM locks for the data it updates. One instance disowns the PCM locks only if the other instance needs access to the same data.

By contrast, if you assign one set of PCM locks to datafiles 3 and 4, I/O increases. This is because both instances regularly use the same set of PCM locks.

Setting GC_FILES_TO_LOCKS: PCM Locks for Each Datafile

Set the GC_FILES_TO_LOCKS initialization parameter to specify the number of PCM locks covering data blocks in a datafile or set of datafiles. This section covers:

- GC_FILES_TO_LOCKS Syntax
- Fixed Lock Examples
- Releasable Lock Example
- Guidelines

Note: Whenever you add or resize a datafile, create a tablespace, or drop a tablespace and its datafiles, adjust the value of GC_FILES_TO_LOCKS before restarting Oracle with OPS enabled.

See Also: Chapter 9, "Parallel Cache Management Instance Locks", to understand how the number of data blocks covered by a single PCM lock is determined.

GC_FILES_TO_LOCKS Syntax

The syntax for setting the GC_FILES_TO_LOCKS parameter specifies the translation between the database address and class of a database block, and the lock name protecting it. You cannot specify this translation for files not mentioned in the GC_FILES_TO_LOCKS parameter.

The syntax for setting this parameter is:

GC_FILES_TO_LOCKS="{*file_list=#locks*[!*blocks*][R][EACH][:]} . . ."

Where:

file_list	*file_list* specifies a single file, range of files, or list of files and ranges as follows: *fileidA*[-*fileidC*][,*fileidE*[-*fileidG*]] ... Query the data dictionary view DBA_DATA_FILES to find the correspondence between file names and file ID numbers.
#locks	Sets the number of PCM locks to assign to file_list. A value of zero (0) for *#locks* means that fine grain locks will be used instead of hashed locks.
!*blocks*	Specifies the number of contiguous data blocks to be covered by each lock.
EACH	Specifies *#locks* as the number of locks to be allocated to *each* file in *file_list*.
R	Specifies that the hashed locks are releasable: they may be released by the instance when no longer needed. Releasable hashed PCM locks are taken from the pool GC_RELEASABLE_LOCKS.

Note: GC_ROLLBACK_LOCKS uses the same syntax.

Spaces are not permitted within the quotation marks of the GC_FILES_TO_LOCKS parameter.

In addition to controlling the mapping of PCM locks to datafiles, GC_FILES_TO_LOCKS controls the number of locks in the default bucket. The default bucket is used for all files not explicitly mentioned in GC_FILES_TO_LOCKS. A value of zero can be used and the default is "0=0". For example, "0=100", "0=100R", "0-9=100EACH". By default, locks in this bucket are releasable; you can however, set these locks to be fixed.

You can specify releasable hashed PCM locks by using the R option with the GC_FILES_TO_LOCKS parameter. Releasable hashed PCM locks are taken from the pool of GC_RELEASABLE_LOCKS

REACH is a keyword that combines "R" and "EACH". For example, GC_FILES_TO_LOCKS="0-9=100REACH". EACHR is *not* a valid keyword.

Omitting EACH and "!*blocks*" means that *#locks* PCM locks are allocated collectively to *file_list* and individual PCM locks cover data blocks for every file in *file_list*. However, if any datafile contains fewer data blocks than the number of PCM locks, some PCM locks will not cover a data block in that datafile.

The default value for *!blocks* is 1. When specified, *blocks* contiguous data blocks are covered by each one of the *#locks* PCM locks. To specify a value for *blocks*, you must use the "!" separator. You would primarily specify *blocks*, and not specify the EACH keyword, to allocate sets of PCM locks to cover multiple datafiles. You can use *blocks* to allocate a set of PCM locks to cover a single datafile where PCM lock contention on that datafile is minimal, thus reducing PCM lock management.

Always set the *!blocks* value to avoid breaking data partitioning gained by using free list groups. Normally you do not need to pre-allocate disk space. When a row is inserted into a table and new extents need to be allocated, contiguous blocks specified with *!blocks* in GC_FILES_TO_LOCKS are allocated to the free list group associated with an instance.

Fixed Lock Examples

For example, you can assign 300 locks to file 1 and 100 locks to file 2 by adding the following line to the parameter file of an instance:

```
GC_FILES_TO_LOCKS = "1=300:2=100"
```

The following entry specifies a total of 1500 locks; 500 each for files 1, 2, and 3:

```
GC_FILES_TO_LOCKS = "1-3=500EACH"
```

By contrast, the following entry specifies a total of only 500 locks spread across the three files:

```
GC_FILES_TO_LOCKS = "1-3=500"
```

The following entry indicates that 1000 distinct locks should be used to protect file 1. The data in the files is protected in groups of 25 blocks.

```
GC_FILES_TO_LOCKS = "1=1000!25"
```

The following entry indicates that the 1000 hashed locks protecting file 1 in groups of 25 blocks may be released by the instance when no longer needed.

```
GC_FILES_TO_LOCKS = "1=1000!25R"
```

Releasable Lock Example

To specify fine grain locks for data blocks with a group factor, specify the following in the parameter file of an instance:

```
GC_FILES_TO_LOCKS="1=0!4"
```

This specifies fine grain locks with a group factor of 4 for file 1.

Guidelines

Use the following guidelines to set the GC_FILES_TO_LOCKS parameter:

- Always specify all datafiles in GC_FILES_TO_LOCKS.
- Assign the same value to GC_FILES_TO_LOCKS for each instance accessing the same database.
- The number of PCM locks you specify for a datafile should never exceed the number of blocks in the datafile. This ensures that if a datafile increases in size, the new blocks can be covered by the extra PCM locks.

  If a datafile is defined with the AUTOEXTEND clause or you issue the ALTER DATABASE ... DATAFILE ... RESIZE command, you should regularly monitor the datafile for an increase in size. If the datafile's size is increasing, update the parameter GC_FILES_TO_LOCKS as soon as possible, then shut down and restart your parallel server.

  Note: Restarting OPS is not required. But if you do not shut down and restart, the locks will cover more blocks.

  If the number of PCM locks specified for *file_list* is less than the actual number of data blocks in the datafiles, then the IDLM uses some PCM locks to cover more datablocks than specified. This can hurt performance, so you should always ensure that sufficient PCM locks are available.
- When you add new datafiles, always specify their locks in GC_FILES_TO_LOCKS to avoid automatic allocation of the "spare" PCM locks.

At some point, you may need to add a datafile using the ALTER TABLESPACE ... ADD DATAFILE command, with OPS running. If you do this, then you should update GC_FILES_TO_LOCKS as soon as possible, then shut down and restart your parallel server.

- To reduce resource contention by creating disjoint data to be accessed by different instances, you should place datafiles on different disks. Use GC_FILES_TO_LOCKS to allocate PCM locks to cover the data blocks in the separate datafiles.
- Specify relatively fewer PCM locks for blocks containing infrequently modified index data. Place indexes in their own tablespace or in their own datafiles within a tablespace so a separate set of PCM locks can be assigned to them. For a read-only index, only one PCM lock is needed.
- Files not mentioned in GC_FILES_TO_LOCKS use DBA fine-grained locking.

Tips for Setting GC_FILES_TO_LOCKS

Setting GC_FILES_TO_LOCKS is an important tuning task in OPS. This section covers some simple checks to help ensure your parameter settings are providing the best performance. This section covers:

- Providing Room for Growth
- Checking for Valid Number of Locks
- Checking for Valid Lock Assignments
- Setting Tablespaces to Read-only
- Checking File Validity
- Adding Datafiles without Changing Parameter Values

Providing Room for Growth

Sites that run continuously cannot afford to shut down to permit adjustment of parameter values. Therefore, when you size these parameters, remember to provide room for growth, or room for files to extend.

Additionally, whenever you add or resize a datafile, create a tablespace, or drop a tablespace and its datafiles, adjust the value of GC_FILES_TO_LOCKS before restarting Oracle with OPS enabled.

Checking for Valid Number of Locks

Check that the number of locks allocated is not larger than the number of data blocks allocated.

> **Note:** Blocks currently allocated may be zero if you are about to insert into a table.

Check the FILE_LOCK data dictionary view to see the number of locks allocated per file. Check V$DATAFILE to see the maximum size of the data file.

> **See Also:** *Oracle8i Reference* for more information about FILE_LOCK and V$DATAFILE.

Checking for Valid Lock Assignments

To avoid lock assignment problems, check the following:

- Do not assign locks to files that only hold rollback segments.
- Do not assign locks to files that only hold temporary data for internal temporary tables.
- Group read-only objects together and assign one lock only to that file. This only works if there is absolutely no writing done to the file or even changes to the blocks, such as block clean out, and so on. If possible, make these tablespaces read-only.

Setting Tablespaces to Read-only

If a tablespace is actually read-only, consider setting it to read-only in Oracle. This ensures that no write to the database occurs and no PCM locks are used. The exception to this is a single lock you can assign to ensure the tablespace will not have to contend for spare locks.

Checking File Validity

Count the number of objects in each file, as follows:

```
SELECT E.FILE_ID      FILE_ID,
       COUNT(DISTINCT OWNER||NAME ) OBJS
  FROM DBA_EXTENTS     E,
       EXT_TO_OBJ V
 WHERE E.FILE_ID = FILE#
   AND E.BLOCK_ID >= LOWB
   AND E.BLOCK_ID <= HIGHB
   AND KIND != 'FREE EXTENT'
   AND KIND != 'UNDO'
 GROUP BY E.FILE_ID;
```

Examine the files storing multiple objects. Run CATPARR.SQL to use the EXT_TO_OBJ view. Make sure the objects can coexist in the same file. That is, make sure the GC_FILES_TO_LOCKS settings are compatible.

Adding Datafiles without Changing Parameter Values

Consider the consequences for PCM lock distribution if you add a datafile to the database. You cannot assign locks to this file without shutting down the instance, changing the GC_FILES_TO_LOCKS parameter, and restarting the database. This may not be possible for a production database.

In this case, the datafile will be assigned to the pool of remaining locks and the file must contend with all files not mentioned in the GC_FILES_TO_LOCKS parameter.

Setting Other GC_* Parameters

This section describes how to set two additional GC_* parameters:

- Setting GC_RELEASABLE_ LOCKS
- Setting GC_ROLLBACK_ LOCKS

Setting GC_RELEASABLE_ LOCKS

For GC_RELEASABLE_LOCKS, Oracle recommends the default setting. This is the value of DB_BLOCK_BUFFERS. This recommendation generally provides optimal performance. However, you can set GC_RELEASABLE_LOCKS to less than the default to save memory, or more than the default to get a possible reduction in locking activity. Too low a value for GC_RELEASABLE_LOCKS could adversely affect performance.

The statistic "releasable freelist waits" in the V$SYSSTAT view tracks the number of times the system runs out of releasable locks. If this condition occurs, as indicated by a non-zero value for releasable freelist waits, you must increase the value of GC_RELEASABLE_LOCKS.

Setting GC_ROLLBACK_ LOCKS

If you are using fixed locks, it is wise to check that the number of locks allocated is not larger than the number of data blocks allocated. Blocks currently allocated may be zero if you are about to insert into a table. Find the number of blocks allocated to a rollback segment by entering:

```
SELECT S.SEGMENT_NAME NAME,
       SUM(RBLOCKS) BLOCKS
  FROM DBA_SEGMENTS S,
       DBA_EXTENTS R
WHERE S.SEGMENT_TYPE = 'ROLLBACK'
  AND S.SEGMENT_NAME = R.SEGMENT_NAME
GROUP BY S.SEGMENT_NAME;
```

This query displays the number of blocks allocated to each rollback segment. When there are many unnecessary forced reads/writes on the undo blocks, try using releasable locks. By default, all rollback segments are protected by releasable locks.

The parameter GC_ROLLBACK_LOCKS takes arguments much like the GC_FILES_TO_LOCKS parameter, for example:

GC_ROLLBACK_LOCKS="0=100:1-10=10EACH:11-20=20EACH"

In this example rollback segment 0, the system rollback segment, has 100 locks. Rollback segments 1 through 10 have 10 locks each, and rollback segments 11 through 20 have 20 locks each.

Note: You cannot use GC_ROLLBACK_LOCKS to make undo segments share locks. The first example below is invalid, but the second is valid, since each of the undo segments has 100 locks to itself:

Invalid: GC_ROLLBACK_LOCKS="1-10=100".

Valid: GC_ROLLBACK_LOCKS="1-10=100EACH".

Tuning PCM Locks

This section discusses several issues to consider before tuning PCM locks:

- Detecting False Pinging
- How Much Time Do PCM Lock Conversions Take?
- Which Sessions Are Waiting for PCM Lock Conversions to Complete?
- What Is the Total Number of PCM Locks and Resources Needed?

Detecting False Pinging

False pinging occurs when you down-convert a lock element protecting two or more blocks that are concurrently updated from different nodes. Assume that each node is updating a different block covered by the same lock. In this event, each node must ping both blocks, even though the node is updating only one of them. This is necessary because the same lock covers both blocks.

No statistics are available to show false pinging activity. To asses false pinging, you can only consider circumstantial evidence. This section describes activity you should look for.

The following SQL statement shows the number of lock operations causing a write, and the number of blocks actually written:

```
SELECT VALUE/(A.COUNTER + B.COUNTER + C.COUNTER) "PING RATE"
  FROM V$SYSSTAT,
    V$LOCK_ACTIVITY A,
    V$LOCK_ACTIVITY B,
    V$LOCK_ACTIVITY C
WHERE A.FROM_VAL = 'X'
    AND A.TO_VAL = 'NULL'
    AND B.FROM_VAL = 'X'
```

```
AND B.TO_VAL = 'S'
AND C.FROM_VAL = 'X'
AND C.TO_VAL = 'SSX'
AND NAME = 'DBWR cross instance writes';
```

Table 15–1 shows how to interpret the ping rate.

Table 15–1 Interpreting the Ping Rate

Ping Rate	Meaning
< 1	False pings may be occurring, but there are more lock operations than writes for pings. DBWR is writing out blocks fast enough, causing no write for a lock activity. This is also known as a "soft ping", meaning I/O activity is not required for the ping, only lock activity.
= 1	Each lock activity involving a potential write causes the write to happen. False pinging may be occurring.
> 1	False pings are definitely occurring.

Use this formula to calculate the percentage of pings that are definitely false:

$$\frac{(ping\_rate - 1)}{ping\_rate} * 100$$

Then check the total number of writes and calculate the number due to false pings:

```
SELECT Y.VALUE "ALL WRITES",
    Z.VALUE "PING WRITES",
    Z.VALUE * pingrate "FALSE PINGS",
FROM V$SYSSTAT Z,
    V$SYSSTAT Y,
WHERE Z.NAME = 'DBWR cross instance writes'
AND Y.NAME = 'physical writes';
```

Here, *ping_rate* is given by the following SQL statement:

```
CREATE OR REPLACE VIEW PING_RATE AS
SELECT ((VALUE/(A.COUNTER+B.COUNTER+C.COUNTER))-1)/
    (VALUE/(A.COUNTER+B.COUNTER+C.COUNTER)) RATE
FROM V$SYSSTAT,
    V$LOCK_ACTIVITY A,
    V$LOCK_ACTIVITY B,
    V$LOCK_ACTIVITY C
WHERE A.FROM_VAL = 'X'
    AND A.TO_VAL   = 'NULL'
    AND B.FROM_VAL = 'X'
    AND B.TO_VAL   = 'S'
    AND C.FROM_VAL = 'X'
AND C.TO_VAL   = 'SSX'
AND NAME = 'DBWR cross instance writes';
```

Needless to say, the goal is not only to reduce overall pinging, but also to reduce false pinging. To do this, look at the distribution of instance locks in GC_FILES_TO_LOCKS and check the data in the files.

How Much Time Do PCM Lock Conversions Take?

Be sure to check the amount of time needed for a PCM lock to convert. This time differs across systems. Enter the following SQL statement to find the lock conversion duration:

```
SELECT *
FROM V$SYSTEM_EVENT
WHERE EVENT = 'lock element cleanup'
```

This SQL statement displays a table similar to the following:

```
                     TOTAL_    TOTAL_      TIME_     AVERAGE_
EVENT                WAITS     TIMEOUTS    WAITED    WAIT
-------------------  ------    -------     ------    ----------
lock element cleanup 32709     44          685660    20.9624262
```

This means that a lock conversion took 20.9 hundredths of a second (0.209 seconds).

Which Sessions Are Waiting for PCM Lock Conversions to Complete?

Enter the following SQL statement to see which sessions are currently waiting, and which have just waited for a PCM lock conversion to complete:

```
SELECT *
FROM V$SESSION_WAIT
WHERE EVENT = 'lock element cleanup'
```

What Is the Total Number of PCM Locks and Resources Needed?

This section explains how to determine the number of PCM locks and resources your system requires. This is the value you need to set for the LM_LOCKS and LM_RESS parameters.

Formula for PCM Locks and Resources

To find this value, add the number of *fixed (non-releasable)* locks set *per instance* (the sum of GC_FILES_TO_LOCKS and GC_ROLLBACK_LOCKS—fixed locks only) to the total number of *releasable* locks (the value of GC_RELEASABLE_LOCKS), and multiply by two.

2 *(GC_FILES_TO_LOCKS + GC_ROLLBACK_LOCKS*fixed* + GC_RELEASABLE_LOCKS)

This figure represents the maximum number of PCM locks and resources your system requires. This calculation is independent of the number of instances.

Also consider the following:

- GC_FILES_TO_LOCKS: the default value is releasable; all instances must have the same setting
- GC_ROLLBACK_LOCKS: the default is releasable; all instances must have the same setting
- GC_RELEASABLE_LOCKS: the default is releasable, set to value of DB_BLOCK_BUFFERS

Calculating PCM Locks and Resources: Example

Assume your system has the following settings *for each instance*:

GC_FILES_TO_LOCKS="1=100:2-5=1000:6-10=1000EACH:11=100R"

GC_ROLLBACK_LOCKS="1-10=10EACH:11-20=20EACH"

GC_RELEASABLE_LOCKS=50,000

Add the GC_FILES_TO_LOCKS values as follows: File 1 has 100 fixed locks. Files 2, 3, 4, and 5 share 1000 locks. File 6 has 1000 fixed locks, file 7 has 1000 fixed locks, file 8 has 1000 fixed locks, file 9 has 1000 fixed locks, and file 10 has 1000 fixed locks. File 11 does not have fixed locks. Hence there is a total of 6,100 fixed locks set by GC_FILES_TO_LOCKS.

Add the GC_ROLLBACK_LOCKS values as follows: Files 1 through 10 have 10 fixed locks each, and Files 11 through 20 have 20 fixed locks each, for a total of 300 fixed locks.

Entering these figures into the formula, calculate the following:

2 * (6,100 + 300 + 50,000) = 112,800

You would thus set the LM_LOCKS and LM_RESS parameters to 112,800.

Note: Use the above equation only to calculate PCM resources.

See Also: For the complete equation to calculate locks and resources, please refer to "Determining the Amount of Locks Needed and Setting LM_* Parameters" on page 18-12. Also refer to "GC_FILES_TO_LOCKS Syntax" on page 15-7 to see details about the syntax of using this parameter.

16

Ensuring IDLM Capacity for Resources and Locks

To reduce contention for shared resources and gain maximum Oracle Parallel Server (OPS) performance, ensure that the Integrated Distributed Lock Manager (IDLM) is adequately configured for all locks and resources your system requires. This chapter covers the following topics:

- Overview
- Planning IDLM Capacity
- Calculating the Number of Non-PCM Resources
- Adjusting Oracle Initialization Parameters
- Minimizing Table Locks to Optimize Performance

> **See Also:** Chapter 10, "Non-PCM Instance Locks" for a conceptual overview.

Overview

Planning PCM locks alone is *not* sufficient to manage locks on your system. Besides explicitly allocating parallel cache management locks, you must actively ensure IDLM is adequately configured, on each node, for all required PCM and non-PCM locks and resources. Consider also that larger databases and higher degrees of parallelism require increased demands for many resources.

Many different types of non-PCM lock exist, and each is handled differently. Although you cannot directly adjust their number, you can estimate the overall number of non-PCM resources and locks required, and adjust the LM_* or GC_*

initialization parameters (or both) to guarantee adequate space. You also have the option of minimizing table locks to optimize performance.

Planning IDLM Capacity

Carefully plan and configure the number of resources and locks to be managed by the IDLM. Allocate these locks and resources using the initialization parameters LM_LOCKS and LM_RESS. Although additional locks and resources can be allocated dynamically, you should avoid this.

Avoiding Dynamic Allocation of Resources and Locks

If the number of locks or resources required becomes greater than the amount you have allocated, additional locks or resources will be allocated from the SGA shared pool. This feature prevents the instance from stopping.

Dynamic allocation causes Oracle to write a message to the alert file indicating that you should recompute and adjust the initialization parameters for the next time the database is started. Since performance and memory usage may be adversely affected by dynamic allocation, it is highly recommended that you correctly compute your lock and resource needs.

Recommended SHARED_POOL_SIZE Settings

The recommended default value for SHARED_POOL_SIZE is 16MB for 64-bit applications, and 8MB for 32-bit applications.

Computing Lock and Resource Needs

Use the following approach to carefully plan IDLM capacity, on a per node basis, for the total number of PCM and non-PCM resources and locks needed.

1. Consider failover requirements.

   In case of failover, you need enough resources configured on the remaining instances so the system can continue operating. Thus, if resources are shared by 10 instances and 5 instances fail, the system must be able to run on the remaining 5 instances. To do this, you must somewhat over estimate system resources by accounting for overhead. In other words, set large enough values for the Oracle initialization parameters determining IDLM locks and resources for each instance.

2. Consider the sizing of each instance on each node: number of users, volume of transactions, and so on. Determine the values you will assign to each instance's initialization parameters.
3. Calculate the number of non-PCM resources and locks required, by filling in the worksheets provided in this chapter.
4. Calculate the number of PCM resources and locks required, by using the script in "What Is the Total Number of PCM Locks and Resources Needed?" on page 15-17.
5. Configure the IDLM to accommodate the required number of:
   - Non-PCM resources
   - Non-PCM locks
   - PCM resources
   - PCM locks

Monitoring Resource Utilization

The V$RESOURCE_LIMIT view provides information about global resource use for some system resources. Using this view to monitor the current and maximum resource use. It is important to notice when the values approach the limits. With this information you can make better decisions when choosing values for resource limit-controlling parameters.

> **See Also:** "Determining the Amount of Locks Needed and Setting LM_* Parameters" on page 18-12. Also refer to the *Oracle8i Reference* regarding V$RESOURCE_LIMIT, and to *Oracle8i Tuning* for a complete discussion of resource limits.

Calculating the Number of Non-PCM Resources

Use the following worksheet to analyze your system resources.

1. In the following worksheet, enter values for the PROCESSES, DML_LOCKS, TRANSACTIONS, and ENQUEUE_RESOURCES initialization parameters for each instance.
2. For each instance, enter the value of the DB_FILES parameter. This is the same for all instances.
3. Enter values for Enqueue Locks for each instance. For each instance, you can calculate this value as follows:

   Enqueue Locks = 20 + (10 * SESSIONS) + DB_FILES + 1 + (2 * PROCESSES) + (DB_BLOCK_BUFFERS / 64)

4. For each instance, enter values for parallel query overhead to cover inter-instance communication. For individual instances, you can calculate this value as follows:

   PQ Overhead = 7 + (MAXINSTANCES * PARALLEL_MAX_SERVERS) + PARALLEL_MAX_SERVERS + MAXINSTANCES

5. Add the entries horizontally to obtain the Subtotals: # of Non-PCM Resources per Instance.
6. Add the per-instance subtotals to obtain the Total Number of Non-PCM Resources System-Wide.

Table 16–1 Worksheet: Calculating Non-PCM Resources

Inst. No.	PRO-CESSES	DML_ LOCKS	TRANS-ACTIONS	ENQUEUE_ RESOURCES	DB_FILES (on one or more instances)	Enqueue Locks	PQ Over head	Over-head	Subtotals: # Non-PCM Resources per Instance
1								200	
2								200	
3								200	
4								200	

Total Number of Non-PCM Resources System-Wide:

7. Finally, use the figures derived from this worksheet to ensure that LM_RESS is set to accommodate all non-PCM resources (see step 6 on on page 16-4).

> **Note:** The worksheet incorporates a standard overhead value of 200 for each instance.

Table 16–2 shows sample values for a system with four instances, and with PARALLEL_MAX_SERVERS set to 8 for instances 1 and 3, and set to 4 for instances 2 and 4. The buffer cache size is assumed to be 10K.

Table 16–2 Calculating Non-PCM Resources (Example)

Inst. No.	PRO-CESSES	DML_ LOCKS	TRANS-ACTIONS	ENQUEUE_ RESOURCES	DB_FILES (on one or more instances)	Enqueue Locks	PQ Over-head	Over head	Subtotals: # Non-PCM Resources per Instance
1	200	500	50	800	30	2808	51	200	4,639
2	350	600	100	1,000	--	4128	31	200	6,409
3	175	400	75	800	--	2453	51	200	4,154
4	225	350	125	1,200	--	3103	31	200	5,234
Total Number of Non-PCM Resources System-Wide:									20,436

Adjusting Oracle Initialization Parameters

Another way to ensure your system has enough space for the required non-PCM locks and resources is to adjust the values of the following Oracle initialization parameters:

- DB_BLOCK_BUFFERS
- DB_FILES
- DML_LOCKS
- PARALLEL_MAX_SERVERS
- PROCESSES
- SESSIONS
- TRANSACTIONS

Begin by experimenting with these values *in the worksheets* supplied in this chapter. You could artificially inflate parameter values in the worksheets to see the IDLM ramifications of providing extra room for failover.

Do not, however, specify actual parameter values considerably greater than needed for each instance. Setting these parameters unnecessarily high entails overhead in a parallel server environment.

Minimizing Table Locks to Optimize Performance

This section describes two strategies for improving performance by minimizing table locks:

- Setting DML_LOCKS to Zero
- Disabling Table Locks

Obtaining table locks (DML locks) for inserts, deletes, and updates can hurt performance in OPS. Locking a table in OPS is very undesirable because all instances holding locks on the table must release those locks. Consider disabling these locks entirely.

Note: If you use either of these strategies, you cannot perform DDL commands against either the instance or the table.

Setting DML_LOCKS to Zero

Table locks are set with the initialization parameter DML_LOCKS. If the DROP TABLE, CREATE INDEX, and LOCK TABLE commands are not needed, set DML_LOCKS to zero to minimize lock conversions and gain maximum performance.

Note: If DML_LOCKS is set to zero on one instance, it must be set to zero on *all* instances. With other values, this parameter need *not* be identical on all instances.

Disabling Table Locks

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

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

Users attempting to lock a table when its table lock is disabled will receive an error.

To re-enable table locking, use the following command:

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

The above command waits until all currently executing transactions commit before enabling the table lock. The command does not need to wait for new transactions starting after issuing the ENABLE command.

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

17

Using Free List Groups to Partition Data

This chapter explains how to allocate free lists and free list groups to partition data. By doing this, you can minimize contention for free space when using multiple instances.

> **Note:** You should only use free list groups to partition data when your application constraints do not allow you to partition the tables. It is much simpler to use partitioned tables and indexes to accomplish the same thing that free list groups accomplish.

The chapter describes:

- Overview
- Deciding How to Partition Free Space for Database Objects
- Setting FREELISTS and FREELIST GROUPS in the CREATE Statement
- Associating Instances, Users, and Locks with Free List Groups
- Pre-allocating Extents (Optional)
- Dynamically Allocating Extents
- Identifying and Deallocating Unused Space

> **See Also:** Chapter 11, "Space Management and Free List Groups" for a conceptual overview and *Oracle8i Concepts*.

Overview

Use the following procedure to manage free space for multiple instances:

1. Analyze your database objects and decide how to partition free space and data.
2. Set FREELISTS and FREELIST GROUPS in the CREATE statement for each table, cluster, and index.
3. Associate instances, users, and locks with free lists.
4. Allocate blocks to free lists.
5. Pre-allocate extents, if desired.

By effectively managing free space, you may improve performance of an application configuration that is not ideally suited to OPS.

Note: For optimal system performance, use care in setting the FREELIST and FREELIST GROUPS options; these values cannot be reset.

Deciding How to Partition Free Space for Database Objects

This section provides a worksheet to help you analyze database objects and decide how to partition free space and data for optimal performance.

- Database Object Characteristics
- Free Space Worksheet

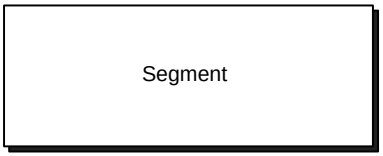
Database Object Characteristics

Analyze the database objects you create and sort them into the categories as described in this section.

Objects in a Static Table

If a table does not have high insert activity, it does not need free lists or free list groups.

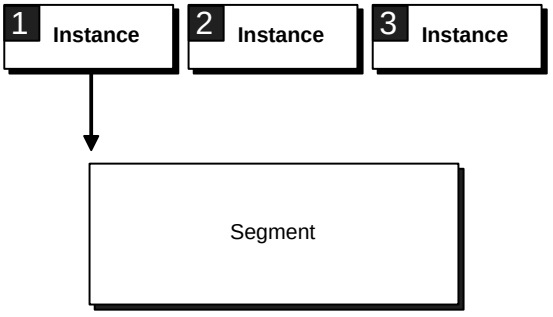
Figure 17–1 Database Objects in a Static Table



Objects in a Partitioned Application

With proper partitioning of certain applications, only one node needs to insert into the table or segment. In such cases, free lists may be necessary if there are a large number of users, but free list groups are not necessary.

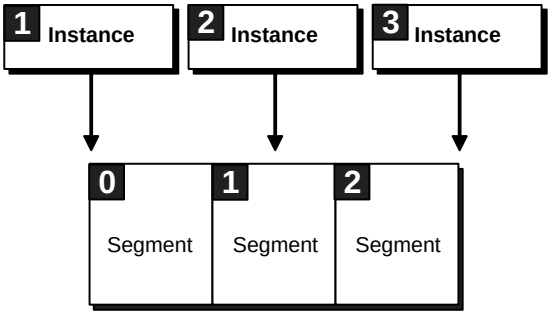
Figure 17–2 Database Objects in a Partitioned Application



Objects Relating to Partitioned Data

Multiple free lists and free list groups are not necessary for objects with partitioned data.

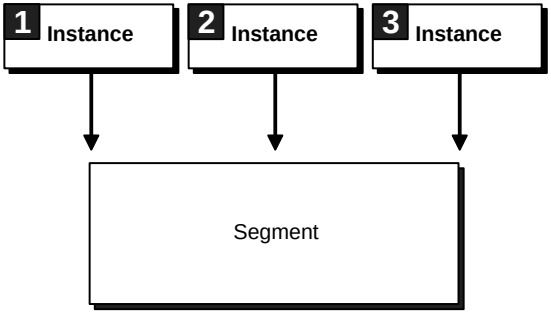
Figure 17–3 Database Objects Relating to Partitioned Data



Objects in a Table with Random Inserts

Free lists and free list groups are needed when random inserts from multiple instances occur in a table. All instances writing to the segment must check the master free list to determine where to write. There would thus be contention for the segment header containing the master free list.

Figure 17–4 Database Objects in a Table with Random Inserts



Free Space Worksheet

List each of your database objects, such as tables, clusters, and indexes, in a worksheet like the following, and plan free lists and free list groups for each.

Table 17–1 Free Space Worksheet for Database Objects

Database Object Characteristics	Free List Groups	Free Lists
Objects in Static Tables	NA	NA
	NA	NA
	NA	NA
	NA	NA
Objects in Partitioned Applications	NA	
	NA	
	NA	
	NA	
Objects Related to Partitioned Data	NA	NA
	NA	NA
	NA	NA
	NA	NA
Objects in Table w/Random Inserts		

Note: Do not confuse partitioned data with Oracle8*i* partitions that may or may not be in use.

Setting FREELISTS and FREELIST GROUPS in the CREATE Statement

This section covers the following topics:

- FREELISTS Option
- FREELIST GROUPS Option
- Creating Free Lists for Clusters
- Creating Free Lists for Indexes

Create free lists and free list groups by specifying the FREELISTS and FREELIST GROUPS storage options in CREATE TABLE, CLUSTER or INDEX statements. You can do this while accessing the database in either exclusive or shared mode.

Note: Once you have set these storage options you *cannot* change their values with the ALTER TABLE, CLUSTER, or INDEX statements.

See Also: The STORAGE clause in Oracle8i SQL Reference for the syntax of these options.

FREELISTS Option

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

FREELIST GROUPS Option

Each free list group is associated with one or more instances at startup. The default value of FREELIST GROUPS is 1, which means that the table's free lists, if any, are available to all instances. Typically, you should set FREELIST GROUPS to the number of instances in OPS. Using free list groups also partitions data. Blocks allocated to one instance, freed by another instance, are no longer available to the first instance.

Note: Even in a non-shared environment, multiple free list groups can improve performance. With multiple free list groups, the free list structure is detached from the segment header, thereby reducing contention for the segment header. This is very useful when there is a high volume of UPDATE and INSERT transactions.

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

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

Creating Free Lists for Clusters

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

Clusters are an optional method of storing data in groups of tables having common columns. Related rows of two or more tables in a cluster are physically stored together within the database to improve access time. OPS allows clusters (other than hash clusters) to use multiple free lists and free list groups.

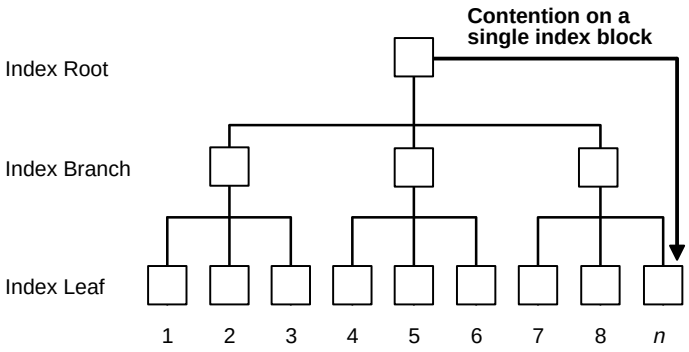
Some hash clusters can also use multiple free lists and free list groups if you created them with a user-defined key for the hashing function and the key is partitioned by instance.

Creating Free Lists for Indexes

You can use the FREELISTS and FREELIST GROUPS storage options of the CREATE INDEX statement to create multiple free space lists for concurrent user processes. Use these options in the same manner as described for tables.

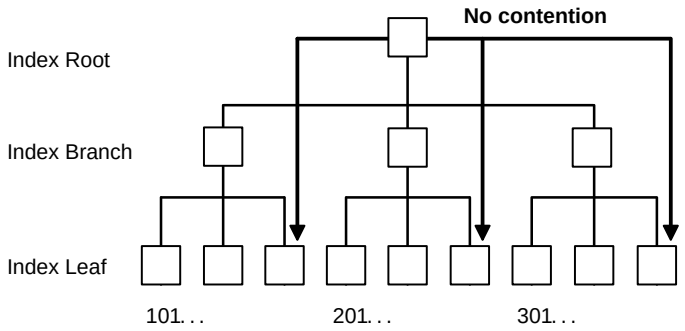
When multiple instances concurrently insert rows into a table having an index, contention for index blocks decreases performance unless index values can be separated by instance. Figure 17–5 illustrates a situation where all instances are trying to insert into the same index leaf block (*n*).

Figure 17–5 Contention for One Index Block



To avoid this problem, have each instance insert into its own tree, as illustrated in Figure 17–6.

Figure 17–6 No Index Contention



Compute the index value with an algorithm such as:

*instance_number * (100000000) + sequence_number*

Associating Instances, Users, and Locks with Free List Groups

This section explains how you can associate the following with free list groups:

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

Associating Instances with Free List Groups

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

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

The SET INSTANCE feature is useful when an instance fails and users connect to other instances. For example, consider a database where space is pre-allocated to the free list groups in a table. With users distributed across instances and the data well partitioned, minimal pinging of data blocks occurs. If an instance fails, moving all users to other instances does not disrupt the data partitioning because each new session can use the original free list group associated with the failed instance.

Associating User Processes with Free List Groups

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

(oracle_pid modulo #free_lists_for_object) + 1

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

Associating PCM Locks with Free List Groups

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

See Also: "Free Lists Associated with Instances, Users, and Locks" on page 11-14.

Pre-allocating Extents (Optional)

This section explains how to pre-allocate extents. This method is useful but a static approach to extent allocation requires a certain amount of database administration overhead.

- The ALLOCATE EXTENT Option
- Setting MAXEXTENTS, MINEXTENTS, and INITIAL Parameters
- Setting the INSTANCE_NUMBER Parameter
- Examples of Extent Pre-allocation

The ALLOCATE EXTENT Option

The ALLOCATE EXTENT option of the ALTER TABLE or ALTER CLUSTER statement enables you to pre-allocate an extent to a table, index or cluster with options to specify the extent size, datafile, and a group of free lists.

The syntax of the ALLOCATE EXTENT option is given in the descriptions of the ALTER TABLE and ALTER CLUSTER statements in *Oracle8 SQL Reference.*

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

The SIZE Option. This option of the ALLOCATE EXTENT clause is the extent size in bytes, rounded up to a multiple of the block size. If you do not specify SIZE, the extent size is calculated according to the values of storage parameters NEXT and PCTINCREASE.

The value of SIZE is *not* used as a basis for calculating subsequent extent allocations, which are determined by NEXT and PCTINCREASE.

The DATAFILE Option. This option specifies the datafile from which to take space for the extent. If you omit this option, space is allocated from any accessible datafile in the tablespace containing the table.

The filename must exactly match the string stored in the control file, even with respect to the case of letters. You can check the DBA_DATA_FILES data dictionary view for this string.

The INSTANCE Option. This option assigns the new space to the free list group associated with instance number *integer*. Each instance acquires a unique instance number at startup that maps it to a group of free lists. The lowest instance number is 1, not 0; the maximum value is operating system specific. The syntax is as follows:

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

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

modulo(*n*,*#_freelistgroups*) + 1

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

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

See Also: "Instance Numbers and Startup Sequence" on page 18-16.

Setting MAXEXTENTS, MINEXTENTS, and INITIAL Parameters

You can prevent automatic allocations by pre-allocating extents to free list groups associated with particular instances, and setting MAXEXTENTS to the current number of extents (pre-allocated extents plus MINEXTENTS). You can minimize the initial allocation when you create the table or cluster by setting MINEXTENTS to 1 (the default) and setting INITIAL to its minimum value (two data blocks, or 10 K for a block size of 2048 bytes).

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

If you expect to increase the number of nodes in your loosely coupled system at a future time, you can allow for additional instances by creating tables or clusters with more free list groups than the current number of instances. You do not have to allocate any space to those free list groups until they are needed. Only the master free list of free blocks has space allocated to it automatically.

For a data block to be associated with a free list group, either it must be brought below PCTUSED by a process running on an instance using that free list group or it must be specifically allocated to that free list group. Therefore, a free list group that is never used does not leave unused free data blocks.

Setting the INSTANCE_NUMBER Parameter

The INSTANCE_NUMBER initialization parameter allows you to start up an instance and ensure that it uses the extents allocated to it for inserts and updates. This will ensure that it does not use space allocated for other instances. The instance cannot use data blocks in another free list unless the instance is restarted with that INSTANCE_NUMBER.

You can also override the instance number during a session by using an ALTER SESSION statement.

Examples of Extent Pre-allocation

This section provides examples in which extents are pre-allocated.

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

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

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

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

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

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

In a parallel server running more instances than the value of the FREELIST GROUPS storage option, multiple instances share the new space allocation. In this example, every third instance to start up is associated with the same group of free lists.

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

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

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

When you need additional space beyond this allocation, use ALTER TABLE to increase MAXEXTENTS before allocating the additional extents. For example, if the

second group of free lists requires additional free space for inserts and updates, you could set MAXEXTENTS to 5 and allocate another extent for that free list group:

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

Dynamically Allocating Extents

This section explains how to 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. It covers:

- Translation of Block Database Address to Lock Name
- !blocks with ALLOCATE EXTENT Syntax

Translation of Block Database Address to Lock Name

As described in the "Allocating PCM Instance Locks" chapter, the syntax for setting the GC_FILES_TO_LOCKS parameter specifies the translation between the database address of a block, and the lock name that will protect it. Briefly, the syntax is:

GC_FILES_TO_LOCKS = "{ *file_list=#locks* [!*blocks*] [EACH] [:] } ..."

The following entry indicates that 1000 distinct lock names should be used to protect the files in this bucket. The data in the files is protected *in groups of 25 blocks*.

```
GC_FILES_TO_LOCKS = "1000!25"
```

!*blocks* with ALLOCATE EXTENT Syntax

Similarly, the !*blocks* parameter enables you to control the number of blocks which are available for use within an extent. (To be available, blocks must be put onto a free list.). You can use !*blocks* to specify the rate at which blocks are allocated within an extent, up to 255 blocks at a time. Thus,

```
GC_FILES_TO_LOCKS = 1000!10
```

means 10 blocks will be made available each time an instance requires the allocation of blocks.

See Also: Chapter 15, "Allocating PCM Instance Locks".

Identifying and Deallocating Unused Space

This section covers:

- How to Determine Unused Space
- Deallocating Unused Space
- Space Freed by Deletions or Updates

How to Determine Unused Space

The DBMS_SPACE package contains procedures by which you can determine the amount of used and unused space in the free list groups in a table. In this way you can determine which instance needs to start allocating space again. The package is created using the DBMSUTIL.SQL script as described in the *Oracle8i Reference*.

Deallocating Unused Space

Unused space you have allocated to an instance using the ALLOCATE EXTENT command *cannot* be deallocated, because it exists below the high water mark.

Unused space *can* be deallocated from the segment, however, if the space exists within an extent that was allocated dynamically above the high water mark. You can use DEALLOCATE UNUSED with ALTER TABLE or ALTER INDEX command in order to trim the segment back to the high water mark.

Space Freed by Deletions or Updates

Blocks freed by deletions or by updates that shrank rows will go to the free list and free list group of the process that deletes them.

Part IV

Oracle Parallel Server System Maintenance Procedures

18

Administering Multiple Instances

Justice is a machine that, when someone has once given it the starting push, rolls on of itself.

John Galsworthy: *Justice. Act II.*

This chapter describes how to administer instances of a parallel server. It includes the following topics:

- Overview
- Oracle Parallel Server Management
- Defining Multiple Instances with Parameter Files
- Setting Initialization Parameters for Multiple Instances
- Determining the Amount of Locks Needed and Setting LM_* Parameters
- Creating Database Objects for Multiple Instances
- Starting Instances
- Specifying Instances
- Shutting Down Instances
- Limiting Instances for Parallel Query

Overview

This chapter explains how to configure and start up instances for OPS using the following procedure:

1. Define multiple instances by setting up parameter files.
2. Set initialization parameters for multiple instances.
3. Determine how many PCM and non-PCM lock you require and set the LM_* parameters.
4. Create database objects for multiple instances.
5. Starting instances.

The details of this procedure appear after a brief explanation of Oracle Parallel Server Management.

> **See Also:** "Starting Up and Shutting Down" in the *Oracle8i Administrator's Guide.*

Oracle Parallel Server Management

Oracle Parallel Server Management (OPSM) is a comprehensive and integrated system management solution for OPS. OPSM allows you to configure and manage multi-instance databases running in heterogeneous environments through an open client-server architecture.

In addition to managing parallel databases, OPSM allows you to schedule jobs, perform event management, monitor performance, and obtain statistics to tune parallel databases.

For more information about OPSM, refer to the *Oracle Parallel Server Management Configuration Guide for UNIX* and the *Oracle Parallel Server Setup and Configuration Guide*. For installation instructions, please refer to your platform-specific installation guide.

Defining Multiple Instances with Parameter Files

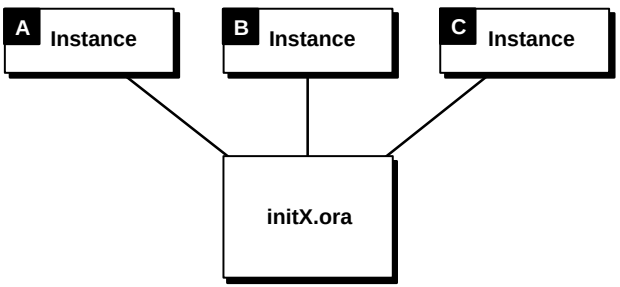
When an instance starts, Oracle uses values in an initialization parameter file to create the System Global Area (SGA) for that instance. You use parameter files in various ways to define multiple instances:

- Using a Common Parameter File for Multiple Instances.
- Using Individual Parameter Files for Multiple Instances.
- Embedding a Parameter File Using IFILE
- Specifying a Non-default Parameter File with PFILE

Using a Common Parameter File for Multiple Instances

A common parameter file for all instances, as shown in Figure 18–1, can simplify administration. If file systems are shared among nodes, you can update all instances by making a change in only one place.

Figure 18–1 Instances with a Common Parameter File

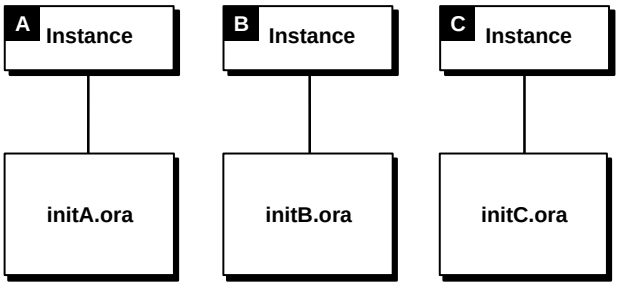


Most clustering systems, however, do not share file systems. In such cases, make a separate physical copy of the common file for each node.

Using Individual Parameter Files for Multiple Instances

Individual parameter files are useful when many parameters differ from instance to instance. For example, initialization parameters to create different sized SGAs for different sized machines may improve performance dramatically.

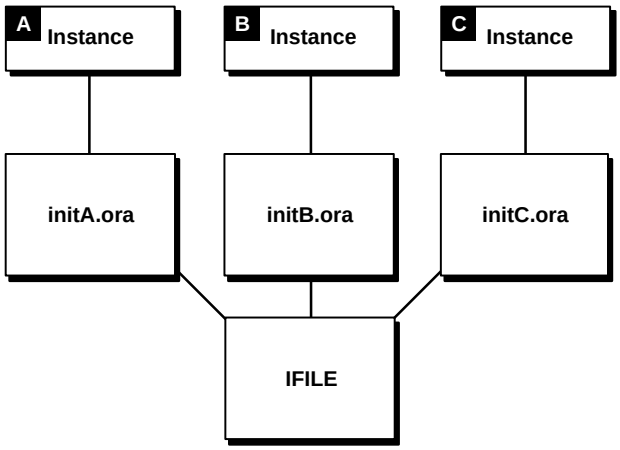
Figure 18–2 Instances with Individual Parameter Files



Embedding a Parameter File Using IFILE

By setting the IFILE parameter, each individual parameter file can embed an additional parameter file containing common values. This approach is illustrated in Figure 18–3.

Figure 18–3 Instances with Individual Parameter Files and IFILE



In Oracle Parallel Server (OPS), some initialization parameters must have the same values for every instance, whether individual or common parameter files are used. By referencing the same file using the IFILE parameter, instances can use their unique parameter files and also ensure they have the correct values for parameters that must be identical across all instances.

Instances *must* use individual parameter files in the following cases:

- Every instance using private rollback segments must have its own parameter file. However, instances only using public rollback segments can all use the same parameter file.
- Every instance specifying an INSTANCE_NUMBER or THREAD parameter must have its own parameter file.

Example

For example, a Server Manager session on the local node can start two instances on remote nodes using individual parameter files named INIT_OPS1.ORA and INIT_OPS2.ORA:

```
SET INSTANCE INSTANCE1;
STARTUP PFILE=INIT_A.ORA;
SET INSTANCE INSTANCE2;
STARTUP PFILE=INIT_B.ORA;
```

Here, "INSTANCE1" and "INSTANCE2" are Net8 aliases for the two respective instances. These aliases are defined in TNSNAMES.ORA.

Both individual parameter files can use the IFILE parameter to include parameter values from the file INIT_COMMON.ORA. They can reference this file as follows:

INIT_OPS1.ORA:

```
IFILE=INIT_COMMON.ORA
INSTANCE_NAME=OPS1
INSTANCE_NUMBER=1
THREAD=1
```

INIT_OPS2.ORA:

```
IFILE=INIT_COMMON.ORA
INSTANCE_NAME=OPS2
INSTANCE_NUMBER=2
THREAD=2
```

Note: Oracle recommends making INSTANCE NAME identical to SID.

The INIT_COMMON.ORA file can contain the following parameter settings that must be identical on both instances.

```
DB_NAME=DB1
SERVICE_NAMES=DB1
CONTROL_FILES=(CTRL_1,CTRL_2,CTRL_3)
GC_FILES_TO_LOCKS="1=600:2-4,9=500EACH:5-8=800"
GC_ROLLBACK_SEGMENTS=10
GC_SEGMENTS=10
LOG_ARCHIVE_START=TRUE
PARALLEL_SERVER=TRUE
```

> **Note:** Oracle recommends that you set SERVICE_NAMES to be identical to DBNAME.

Each parameter file must contain identical values for the CONTROL_FILES parameter, for example, because all instances share the control files.

To change the value of a common initialization parameter, modify the INIT_COMMON.ORA file rather than changing both individual parameter files.

IFILE Use

When you specify parameters having identical values in a common parameter file referred to by IFILE, you can omit parameters for which you are using the default values.

If you use multiple Server Manager sessions on separate nodes to start up the instances, each node must have its own copy of the common parameter file unless the file systems are shared.

If a parameter is duplicated in an instance-specific file and in the common file, or within one file, the last value specified overrides earlier values. You can therefore ensure the use of common parameter values by placing the IFILE parameter at the end of an individual parameter file. Placing IFILE at the beginning of the individual file allows you to override the common values.

You can specify IFILE more than once in a parameter file to include multiple common parameter files. Unlike the other initialization parameters, IFILE does not override previous values. For example, an individual parameter file might include an INIT_COMMON.ORA file and separate command files for the LOG_* and GC_* parameters:

```
IFILE=INIT_COMMON.ORA
IFILE=INIT_LOG.ORA
IFILE=INIT_GC.ORA
LOG_ARCHIVE_START=FALSE
THREAD=3
ROLLBACK_SEGMENTS=(RB_C1,RB_C2,RB_C3)
```

The individual value of LOG_ARCHIVE_START overrides the value specified in INIT_LOG.ORA, because the statement IFILE = INIT_LOG.ORA appears before the LOG_ARCHIVE_START parameter specification. The individual GC_* values specified in INIT_GC.ORA override values specified in INIT_COMMON.ORA because IFILE = INIT_GC.ORA comes after IFILE = INIT_COMMON.ORA.

> **See Also:** "Instance Numbers and Startup Sequence" on page 18-16, "Redo Log Files" on page 6-3, and "Parameters that Must Be Identical on All Instances" on page 18-11.

Specifying a Non-default Parameter File with PFILE

Use the PFILE option of the STARTUP command to specify a parameter file other than the default file when you start up an instance. The parameter file specified by PFILE must be on a disk accessible to the local node, even for an instance on a remote node.

Setting Initialization Parameters for Multiple Instances

This section discusses important OPS initialization parameters for multiple instances.

- GC_* Global Cache Parameters
- Parameter Notes for Multiple Instances
- Parameters that Must Be Identical on All Instances

> **See Also:** *Oracle8i Reference* for details about other Oracle initialization parameters.

GC_* Global Cache Parameters

Initialization parameters with the prefix GC (Global Cache) are relevant only for OPS. The settings of these parameters determine the size of the collection of global locks that protect the database buffers on all instances. The settings you choose affect use of certain operating system resources.

The first instance to start up in shared mode determines the values of the global cache parameters for all instances. The control file records the values of the GC_* parameters when the first instance starts up.

When another instance attempts to start up in shared mode, Oracle compares the values of the global cache parameters in its parameter file with those already in use and issues a message if any values are incompatible. The instance cannot mount the database unless it has correct values for its global cache parameters.

The global cache parameters for OPS are:

GC_FILES_TO_LOCKS	Controls data block locks.
GC_ROLLBACK_LOCKS	Controls undo block locks.
GC_RELEASABLE_LOCKS	Controls the number of locks.

See Also: Chapter 15, "Allocating PCM Instance Locks".

Parameter Notes for Multiple Instances

Table 18–1 summarizes multi-instance issues concerning initialization parameters.

Table 18–1 Initialization Parameter Notes for Multiple Instances

Parameter	Parallel Server Notes
CHECKPOINT_PROCESS	In OPS, your database may have more datafiles than in a single-instance environment. To speed up checkpoints, enable the CHECKPOINT_PROCESS parameter.
DELAYED_LOGGING_BLOCK_ CLEANOUTS	The default value TRUE reduces pinging between instances.
DML_LOCKS	Must be identical on all instances only if set to zero.
INSTANCE_NUMBER	If specified, this parameter must have unique values for different instances.
LOG_ARCHIVE_FORMAT	You must include thread number.
MAX_COMMIT_PROPAGATION_ DELAY	If you want commits to be seen immediately on remote instances, you may need to change the value of this parameter.
NLS_* parameters	Can have different values for different instances.
PARALLEL_SERVER	To enable OPS, this parameter must be set to TRUE in the initialization file. It defaults to FALSE.
PROCESSES	This parameter must have a value large enough to accommodate all background and user processes. Some operating systems can have additional DBWR processes. Defaults for the SESSIONS and TRANSACTIONS parameters are derived directly or indirectly from the value of the PROCESSES parameter. If you do not use defaults, you may want to increase the values for some of the above parameters to allow for LCK*n* and other optional background processes.
RECOVERY_PARALLELISM	To speed up the roll forward or cache recovery phase, you may want to set this parameter.
ROLLBACK_SEGMENTS	Specify the private rollback segments for each instance.
THREAD	If specified, this parameter must have unique values for different instances.

See Also: *Oracle8i Reference* for details about each parameter.

Parameters that Must Be Identical on All Instances

Certain initialization parameters that are critical at database creation or that affect certain database operations must have the same value for every instance in OPS. For example, the values of DB_BLOCK_SIZE and CONTROL_FILES must be identical for every instance. Other parameters can have different values for different instances. The following initialization parameters must have identical values for every instance in a parallel server:

CONTROL_FILES
CPU_COUNT
DB_BLOCK_SIZE
DB_FILES
DB_NAME
DB_DOMAIN
SERVICE_NAMES
DML_LOCKS (must be identical only if set to zero)
GC_FILES_TO_LOCKS
GC_ROLLBACK_LOCKS
LM_LOCKS (identical values recommended)
LM_PROCS (identical values recommended)
LM_RESS (identical values recommended)
MAX_COMMIT_PROPAGATION_DELAY
PARALLEL_DEFAULT_MAX_SCANS
ROLLBACK_SEGMENTS
ROW_LOCKING

See Also: *Oracle8i Reference* for details about each parameter.

Determining the Amount of Locks Needed and Setting LM_* Parameters

Set values for the LM_* initialization parameters. Resources, locks and process configurations are per OPS instance. For ease of administration, these parameters should be consistent for all instances.

LM_RESS	This parameter controls the number of resources that can be locked by the Integrated Distributed Lock Manager (IDLM). This parameter includes non-PCM resources such as the number of lock resources allocated for DML, DDL (data dictionary locks), and data dictionary cache locks plus file and log management locks. Derive a value for LM_RESS by adding the number of PCM and non-PCM resources as calculated in Chapter 15, "Allocating PCM Instance Locks" and Chapter 16, "Ensuring IDLM Capacity for Resources and Locks" respectively.
LM_LOCKS	Number of locks. Where N is the total number of nodes: LM_LOCKS = LM_RESS + (LM_RESS * (N - 1)) / N
LM_PROCS	Number of processes. The value of the PROCESSES initialization parameter multiplied by the number of nodes.

Used increased values if you plan to use parallel DML or DML performed on partitioned objects.

Creating Database Objects for Multiple Instances

Creating a database automatically starts a single instance with OPS disabled. Before you can start multiple instances, however, you must perform certain administrative operations. These tasks may include:

- Creating extra rollback segments for each additional instance,
- Adding and enabling a thread for each additional instance, and
- Providing locks for added datafiles.

You can perform these operations with a single instance in either exclusive or shared mode.

> **See Also:** "Creating Additional Rollback Segments" on page 14-5, "Redo Log Files" on page 6-3, and "What Is the Total Number of PCM Locks and Resources Needed?" on page 15-17.

Starting Instances

An Oracle instance can start with OPS enabled or disabled. This section explains the procedures on how to do this:

- Enabling Parallel Server and Starting Instances
- Starting with OPS Disabled
- Starting in Shared Mode

Enabling Parallel Server and Starting Instances

Note: In Oracle8 the keywords SHARED, EXCLUSIVE, and PARALLEL are obsolete in the STARTUP and ALTER DATABASE MOUNT statements.

Starting an Instance Using SQL

1. To enable OPS in Oracle8, set the PARALLEL_SERVER parameter to TRUE in the initialization file. It defaults to FALSE.
2. Start any required operating system specific processes.

   For more information about these, please see your Oracle system-specific documentation.
3. Ensure your Cluster Manager software is running.

   See "The Cluster Manager" on page 18-22 for more information.
4. Connect with SYSDBA or SYSOPER privileges.

   ```
   CONNECT username/password AS SYSDBA
   ```
5. Make sure the PARALLEL_SERVER initialization parameter is set to TRUE if you wish to run with OPS enabled, or to FALSE to run with OPS disabled.
6. Start an instance.

   ```
   STARTUP NOMOUNT
   ```
7. Mount a database.

   ```
   ALTER DATABASE database_name MOUNT
   ```

8. Open the database.

```
ALTER DATABASE OPEN
```

Starting an Instance Using Server Manager

> **Note:** The Server Manager command STARTUP with the OPEN option performs steps 4, 5, and 6 of the procedure given above.

1. Start any required operating system specific processes.

   For more information, please see your Oracle system-specific documentation.

2. Set the PARALLEL_SERVER initialization parameter to TRUE to run with OPS enabled, or to FALSE to run with OPS disabled.

3. Ensure your Cluster Manager software is running.

   See "The Cluster Manager" on page 18-22 for more information.

4. Start Server Manager.

5. Start up an instance with the OPEN option:

```
STARTUP OPEN database_name
```

Starting with OPS Disabled

OPS must be disabled whenever you change the archiving mode (ARCHIVELOG or NOARCHIVELOG). To change the archiving mode, the database must be mounted but not open.

If an instance mounts a database with PARALLEL_SERVER set to FALSE, no other instance can mount the database.

Before you can start an instance in exclusive mode, shut down all instances running in shared mode. A single instance running in shared mode is not the same as an instance running in exclusive mode, and the last instance running in shared mode does not automatically revert to exclusive mode.

An instance starting with OPS disabled can specify an instance number with the INSTANCE_NUMBER parameter. This is only necessary if the instance performs inserts and updates and if the tables in your database use the FREELIST GROUPS storage option to allocate free space to instances. If you start an instance just to

perform administrative operations with OPS disabled, omit the INSTANCE_NUMBER parameter from the parameter file.

An instance starting with OPS disabled can also specify a thread other than 1 to use the online redo log files associated with that thread.

See Also: Chapter 17, "Using Free List Groups to Partition Data".

Starting in Shared Mode

In OPS, each instance must mount the database in shared mode. Each initialization parameter file for each instance must have the SINGLE_PROCESS parameter set to FALSE and the PARALLEL_SERVER parameter set to TRUE. Before you start up multiple instances in shared mode, create at least one rollback segment for each instance sharing the same database and enable a thread containing at least two groups of redo log files for each additional instance.

If one instance mounts a database in shared mode, other instances can also mount the database in shared mode, but not in exclusive mode.

If PARALLEL_SERVER is set to FALSE, the instance tries to start with OPS disabled by default.

Retrying to Mount a Database in Shared Mode

If you attempt to start an instance and mount a database in shared mode while another instance is currently recovering the same database, your new instance cannot mount the database until the recovery is complete.

Rather than repeatedly attempting to start the instance, you can use the STARTUP RETRY statement. This causes the new instance to retry every five seconds to mount the database until it succeeds or has reached the retry limit. For example:

```
STARTUP OPEN MAILDB RETRY
```

To set the maximum number of times the instance attempts to mount the database, use the Server Manager SET command with the RETRY option; you can specify either an integer (such as 10) or the keyword INFINITE.

If the database can only be opened by being recovered by another instance, then the RETRY will not repeat. For example, if the database was mounted in exclusive mode by one instance, then trying the STARTUP RETRY command in shared mode does not work for another instance.

Instance Numbers and Startup Sequence

When an instance starts up, it acquires an instance number that maps the instance to one group of free lists for each table created with the FREELIST GROUPS storage option.

An instance can specify its instance number explicitly by using the initialization parameter INSTANCE_NUMBER when it starts up with OPS enabled or disabled. If an instance does not specify the INSTANCE_NUMBER parameter, it automatically acquires the lowest available number.

Startup order determines the instance numbers for instances that do not specify the INSTANCE_NUMBER parameter. Startup numbers are difficult to control if instances start up in parallel, and they can change after instances shut down and restart.

Instances using the INSTANCE_NUMBER parameter must specify different numbers. The Server Manager command SHOW PARAMETERS INSTANCE_NUMBER shows the current instance number each instance is using. This command displays a null value if an instance number was assigned based on startup order.

After an instance shuts down, its instance number becomes available again. If a second instance starts up before the first instance restarts, the second instance can acquire the instance number previously used by the first instance.

Instance numbers based on startup order are independent of instance numbers specified with the INSTANCE_NUMBER parameter. After an instance acquires an instance number by one of these methods, either with or without INSTANCE_NUMBER, another instance cannot acquire the same number by the other method. All numbers are unique, regardless of the method by which they are acquired.

Always use the INSTANCE_NUMBER parameter if you need a consistent allocation of extents to instances for inserts and updates. This allows you to maintain data partitioning among instances.

> **See Also:** "Rollback Segments" on page 6-7, "Creating Additional Rollback Segments" on page 14-5, "Redo Log Files" on page 6-3, and Chapter 17, "Using Free List Groups to Partition Data" for information about allocating free space for inserts and updates.

Specifying Instances

When performing administrative operations in a multi-instance environment, be sure you specify the correct instance. This section includes the following topics related to instance-specific administration:

- Differentiating Between Current and Default Instance
- How SQL Statements Apply to Instances
- How Server Manager Commands Apply to Instances
- Specifying Instance Groups
- Using a Password File to Authenticate Users on Multiple Instances

Differentiating Between Current and Default Instance

Some Server Manager commands apply to the instance to which Server Manager is currently connected and others apply to the default instance.

Default instance.	The *default instance* is on the machine where you initiate Server Manager. Server Manager commands that *cannot* be used while you are connected to an instance, such as executing a host command, apply to the default instance.
Current instance.	The *current instance* is determined by the CONNECT command. Server Manager commands that *can* be used while you are connected to an instance apply to the current instance.

The current instance can be different from the default instance if you specify a connect string in the CONNECT command.

Net8 must be installed to use the SET INSTANCE or CONNECT command for an instance running on a remote node.

> **See Also:** Your platform-specific Oracle documentation, for more information about installing Net8 and the exact format required for the connect string used in the SET INSTANCE and CONNECT commands.

How SQL Statements Apply to Instances

Instance-specific SQL statements apply to the current instance. For example, the statement ALTER DATABASE ADD LOGFILE only applies to the instance to which you are currently connected, rather than the default instance or all instances.

ALTER SYSTEM CHECKPOINT LOCAL applies to the current instance. By contrast, ALTER SYSTEM CHECKPOINT or ALTER SYSTEM CHECKPOINT GLOBAL applies to *all* instances.

ALTER SYSTEM SWITCH LOGFILE applies only to the current instance. To force a global log switch, you can use ALTER SYSTEM ARCHIVE LOG CURRENT. The THREAD option of ALTER SYSTEM ARCHIVE LOG allows you to archive online redo log files for a specific instance.

How Server Manager Commands Apply to Instances

When you initiate Server Manager, the commands you enter are relevant to the default instance, which is also the current instance.

This is true until you use the SET INSTANCE command to set the current instance. From that point, all Server Manager commands operate on the current instance.

Table 18–2 describes how these commands relate to instances.

Table 18–2 How Server Manager Commands Apply to Instances

Server Manager Command	Associated Instance
ARCHIVE LOG	Always applies to the current instance.
CONNECT	Uses the default instance if no instance is specified in the CONNECT command.
HOST	Applies to the node running the Server Manager session, regardless of the location of the current and default instances.
MONITOR	MONITOR display screens identify the current instance, not the default instance, in the upper left corner.
RECOVER	Does not apply to any particular instance, but rather to the database.
SHOW INSTANCE	Displays information about the default instance, which can be different from the current instance.
SHOW PARAMETERS	Displays information about the current instance.
SHOW SGA	Displays information about the current instance.
SHUTDOWN	Always applies to the current instance. A privileged Server Manager command.
STARTUP	Always applies to the current instance. A privileged Server Manager command.

Note: The security mechanism invoked when you use privileged Server Manager commands depends on your operating system. Most operating systems have a secure authentication mechanism when logging onto the operating system. On these systems, your default operating system privileges usually determine whether you can use STARTUP and SHUTDOWN. For more information, see your Oracle system-specific documentation.

The SET INSTANCE and SHOW INSTANCE Commands

You can change the default instance with the Server Manager statement:

```
SET INSTANCE instance_path
```

where *instance_path* is a valid Net8 connect string without a user ID / password. If you are connected to an instance, you must disconnect before using SET

INSTANCE. Alternatively, if you do not wish to disconnect from the current instance, you may use the CONNECT command with *instance_path*.

You can use the SET INSTANCE command to specify an instance on a remote node for the commands STARTUP and SHUTDOWN. The parameter file for a remote instance must be on the local node.

The SHOW INSTANCE command displays the connect string for the default instance. SHOW INSTANCE returns the value *local* if you have not used SET INSTANCE during the Server Manager session.

To reset to the default instance, use SET INSTANCE without specifying a connect string or specify LOCAL (but not DEFAULT, which would indicate a connect string for an instance named "DEFAULT").

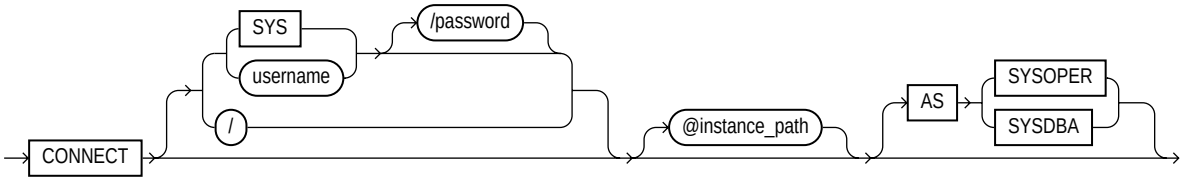
The following Server Manager line mode examples illustrate the relationship between SHOW INSTANCE and SET INSTANCE:

```
SHOW INSTANCE
INSTANCE                       LOCAL

SET INSTANCE node1
Oracle8 Server Release 8.1 - Production
With the distributed, parallel query and Parallel Server options
PL/SQL V8.1 - Production

SHOW INSTANCE
Instance                       node2

SET INSTANCE
ORACLE8 Server Release 8.1 - Production
With the procedural, distributed, and Parallel Server options
PL/SQL V8.1 - Production

SHOW INSTANCE
INSTANCE                       LOCAL

SET INSTANCE DEFAULT
ORA-06030: NETDNT: connect failed, unrecognized node name
```

The CONNECT Command

The CONNECT command associates Server Manager with either the default instance or an instance you explicitly specify. The instance to which Server Manager connects becomes the *current instance*.

The CONNECT command has the following syntax:



where *instance-path* is a valid Net8 connect string. CONNECT without the argument *@instance-path* connects to the default instance (which may have been set previously with SET INSTANCE).

Connecting as SYSOPER or SYSDBA allows you to perform privileged operations, such as instance startup and shutdown.

Multiple Server Manager sessions can connect to the same instance at the same time. When you are connected to one instance, you can connect to a different instance without using the DISCONNECT command. Server Manager disconnects you from the first instance automatically whenever you connect to another one.

The CONNECT *@instance-path* command allows you to specify an instance before using the Server Manager commands MONITOR, STARTUP, SHUTDOWN, SHOW SGA, and SHOW PARAMETERS.

> **See Also:** *Oracle Server Manager User's Guide* for syntax of Server Manager commands, the *Net8 Administrator's Guide* for the proper specification of *instance_path*, and the *Oracle8i Administrator's Guide* for information on connecting with SYSDBA or SYSOPER privileges.

The Cluster Manager

To achieve high availability, OPS cooperates with a platform-specific software component known as the Cluster Manager (CM). Hardware vendors usually provide this component.

CM monitors the status of various resources in a cluster including nodes, interconnect hardware and software, shared disks, and Oracle instances. CM automatically starts and stops when the instance starts and stops.

The CM informs clients and the Oracle server when the statuses of resources change. The Oracle server must know when another database instance registers with the CM or disconnects from it. Database instances register with the CM during the mount phase of startup.

A CM disconnect occurs for any of three reasons: the client disconnects voluntarily, the client's process terminates, or the client's node shuts down or crashes. An important feature of CM is that it provides a global view of the cluster, even during failures. This ensures the integrity of OPS databases, as each instance must be aware of all other instances to coordinate access to shared disks.

Oracle accesses CM through an interface called the "Node Monitor API". This interface provides an abstract link to process groups whose members may be arbitrarily distributed throughout the cluster. When an OPS instance is mounted, the LMON process joins one of these groups. Any instances of the same database that were already running are informed of the new OPS instance. All instances then synchronize to ensure they have the same view of active instances. The CM then informs the IDLM layer about any new instances; the CM also initiates an IDLM reconfiguration. If an instance shuts down or terminates abnormally, CM informs the remaining instances. Again, the CM synchronizes the instances informs the IDLM.

OPS Cluster Administration

When starting an OPS instance, first ensure your CM software is running. Detailed instructions on CM administration appear in platform-specific documentation. If the CM is not available or if Oracle has a problem communicating with it, Oracle displays error ORA-29701: "Unable to connect to Cluster Manager".

Multiple Version Compatibility on Clusters

As long as your Oracle version numbers are greater than 8.1, they can co-exist on the same cluster. This also means you cannot have different versions of Oracle older

than 8.1 on the same cluster. For example, an 8.0 and an 8.1 OPS database are not compatible on the same cluster.

Specifying Instance Groups

For ease of administration, logically group different instances and perform parallel operations on all associated instances at once. You can define an *instance group* as a set of instances used for a specific purpose, such as resource allocation, parallel query or other parallel operations. They thus enable you to partition your resources effectively.

Sometimes, for example, a DBA may wish to prevent users or query processes from obtaining resources on all instances. The DBA may want to keep certain instances available only for users running OLTP processes, and restrict users running parallel queries only to a particular set of instances.

For example, you might create instance groups such that between 9 AM and 5 PM users can use group B, but after 5 PM they can use group D. Or, you might use group C for normal OLTP inserts and updates but use group D for large parallel tasks to avoid interfering with OLTP performance.

- Define all potentially desirable group configurations during startup since you cannot add and delete instances from groups dynamically while the instance is up.
- One instance can be a member of more than one group at any given time. Groups may overlap one another.
- You can define as many groups as you wish, but only use them as needed. Instance groups do not incur much overhead and you are not required to refer to them, once they are defined.

If you simply set the degree of parallelism, the system chooses which specific instances to use given disk affinity, and the number of instances actually running. By specifying instance groups, you can directly specify the instances for parallel operations.

The instance from which you initiate a query need not be a member of the group of instances that perform the query. The parallel coordinator does run on the current instance.

How to Specify Instance Groups

To specify instance groups, set the INSTANCE_GROUPS initialization parameter within the parameter file of each instance you wish to associate to the group. This parameter at once defines a group and adds the current instance to the group.

For example, instance 1 could set the parameter as follows:

```
INSTANCE_GROUPS = GROUPB, GROUPD
```

Instance 3 could set it as follows:

```
INSTANCE_GROUPS = GROUPA, GROUPD
```

As a result, instances 1 and 3 would both belong to instance group D, but would also belong to other groups as well.

You cannot dynamically change INSTANCE_GROUPS.

How to Use Instance Groups

You can use instance groups to identify a group to be used for a parallel operation with PARALLEL_INSTANCE_GROUP

The default for PARALLEL_INSTANCE_GROUP is a group consisting of all currently running instances.

To use a particular instance group for a given parallel operation, specify the following parameter in the initialization parameter file:

```
PARALLEL_INSTANCE_GROUP = GROUPNAME
```

All parallel operations initiated from that instance spawn processes only within that group, using the same algorithm as before either randomly or with disk affinity.

PARALLEL_INSTANCE_GROUP is a dynamic parameter that you change using an ALTER SESSION or ALTER SYSTEM statement. You can use it to refer to only *one* instance group; by default it is set to a default group that includes all currently active instances. The instance upon which you are running need not be a part of the instance group you are going to use for a particular operation.

> **See Also:** The *Oracle8i Reference* for complete information about initialization parameters and views.

How to List Members of Instance Groups

To see the members of different instance groups, query the GV$ global dynamic performance view GV$PARAMETER. Look at all entries for the INSTANCE_GROUPS parameter name.

Instance Group Example

In this example, instance 1 has the following settings in its initialization parameter file:

```
INSTANCE_GROUPS = GA, GB
PARALLEL_INSTANCE_GROUP  GB
```

Instance 2 has the following settings in its initialization parameter file:

```
INSTANCE_GROUPS = GB, GC
PARALLEL_INSTANCE_GROUP = GC
```

On instance 1, if you enter the following statements, the instances in Gb is used. Two server processes spawn on instance 1, and 2 server processes on instance 2.

```
ALTER TABLE TABLE PARALLEL (DEGREE 2 INSTANCES 2);
SELECT COUNT(*) FROM TABLE;
```

If you enter the following statements on instance 1, Gc will be used. Two server processes will be spawned on instance 2 only.

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = 'GC';
SELECT COUNT (*) FROM TABLE;
```

If you enter the following statements on instance 1, the default instance group (all currently running instances) is used. Two server processes spawn on instance 1, and 2 server processes on instance 2.

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = '';
SELECT COUNT(*) FROM TABLE;
```

Using a Password File to Authenticate Users on Multiple Instances

You can use a password file to authenticate users performing database administration when running multiple instances on OPS. In this case, the environment variable for each instance must point to the same password file. Similarly, the REMOTE_LOGIN_PASSWORDFILE initialization parameter for each instance must be set to the appropriate, identical value.

> **See Also:** The *Oracle8i Administrator's Guide* for information about the REMOTE_LOGIN_PASSWORDFILE parameter. For more information on the exact name of the password file, or for the name of the environment variable used to specify this name for your operating system, see your Oracle system-specific documentation.

Shutting Down Instances

Use the following procedure to shut down an instance:

1. Connect with SYSDBA.

   ```
   CONNECT username/password AS SYSDBA
   ```

2. Close the database.

   ```
   ALTER DATABASE database_name CLOSE
   ```

3. Dismount the database.

   ```
   ALTER DATABASE database_name DISMOUNT
   ```

Alternatively, you can use the Server Manager command SHUTDOWN, which performs all three of these steps for the current instance.

In OPS, shutting down one instance does not interfere with the operations of any instances still running.

To shut down a database mounted in shared mode, shut down every instance in the parallel server. OPS allows you to shut down instances in parallel from different nodes. When an instance shuts down abnormally, Oracle forces all user processes running in that instance to log off the database. If a user process is currently accessing the database, Oracle terminates that access and returns the message "ORA-1092: Oracle instance terminated. Disconnection forced". If a user process is not currently accessing the database when the instance shuts down, Oracle returns the message "ORA-1012: Not logged on" upon the next call or request made to Oracle.

After a NORMAL or IMMEDIATE shutdown, instance recovery is not required. Recovery is required, however, after the SHUTDOWN ABORT command or after an instance terminates abnormally. The SMON process of an instance that is still running performs instance recovery for the instance that shut down. If no other instances are running, the next instance to open the database performs instance recovery for any instances that need it.

If multiple Server Manager sessions are connected to the same instance simultaneously, all but one must disconnect before the instance can be shut down normally. You can use the IMMEDIATE or ABORT option of the SHUTDOWN command to shut down an instance when multiple Server Manager sessions (or any other sessions) are connected to it.

> **See Also:** "Starting Up and Shutting Down" in *Oracle8i Administrator's Guide* for options of the SHUTDOWN command.

Limiting Instances for Parallel Query

Although the parallel query feature does not require OPS, some aspects of parallel query apply only to a parallel server.

The INSTANCES keyword of the PARALLEL clause of the CREATE TABLE, ALTER TABLE, CREATE CLUSTER, and ALTER CLUSTER commands allows you to specify that a table or cluster is split up among the buffer caches of all available instances of OPS when the table is scanned in a parallel query.

If you do not want tables to be dynamically partitioned among all available instances, specify the number of instances that can participate in scanning or caching with the ALTER SYSTEM command.

To specify the number of instances to participate in parallel query processing at startup time, specify a value for the initialization parameter PARALLEL_MAX_INSTANCES.

To dynamically limit the number of instances available for parallel query processing, use the ALTER SYSTEM command. For example, if your parallel server has ten instances running but you want only eight to be involved in parallel query processing, while the remaining two instances are dedicated for other use issue the following command:

```
ALTER SYSTEM SET SCAN_INSTANCES = 8;
```

Thereafter, if a table's definition has a value of ten specified for the INSTANCES keyword, the table is scanned by query servers on only eight of the ten instances.

Oracle selects the first eight instances in this example. You can set the PARALLEL_MAX_SERVERS initialization parameter to zero on instances that you do not want to have participating in parallel query processing.

If you wish to limit the number of instances that cache a table, issue the following command:

```
ALTER SYSTEM SET CACHE_INSTANCES = 8;
```

Thereafter, if a table definition has 10 specified for the INSTANCES keyword and the CACHE keyword was specified, the table is divided evenly among eight of the ten available instances' buffer caches.

> **See Also:** "Specifying Instance Groups" on page 18-23 and the *Oracle8i Reference* for more information about parameters. For more information on parallel query or parallel execution, please refer to *Oracle8i Tuning*.

PARALLEL_SERVER_INSTANCES

The parameter PARALLEL_SERVER_INSTANCES specifies the number of instances configured in an OPS environment. Use this parameter to provide information to Oracle to manage the size of SGA structures that depend on the number of instances. When you set this parameter, the system makes better use of available memory.

Instance Registration and Client/Service Connections

Instance registration involves three separate processes. These processes are registration of:

- Service handlers by PMON
- Dedicated server handlers
- Instance registration

PMON performs service handler registration by registering MTS dispatchers with TNS listeners. PMON also registers dedicated server handlers by dynamically communicating with the TNS listener. PMON can also dynamically register dedicated server handlers by telling the TNS listener how to start new dedicated processes.

Registering dedicated server handlers is where TNS listeners of an instance only create dedicated servers if the instance has been registered. This is part of the

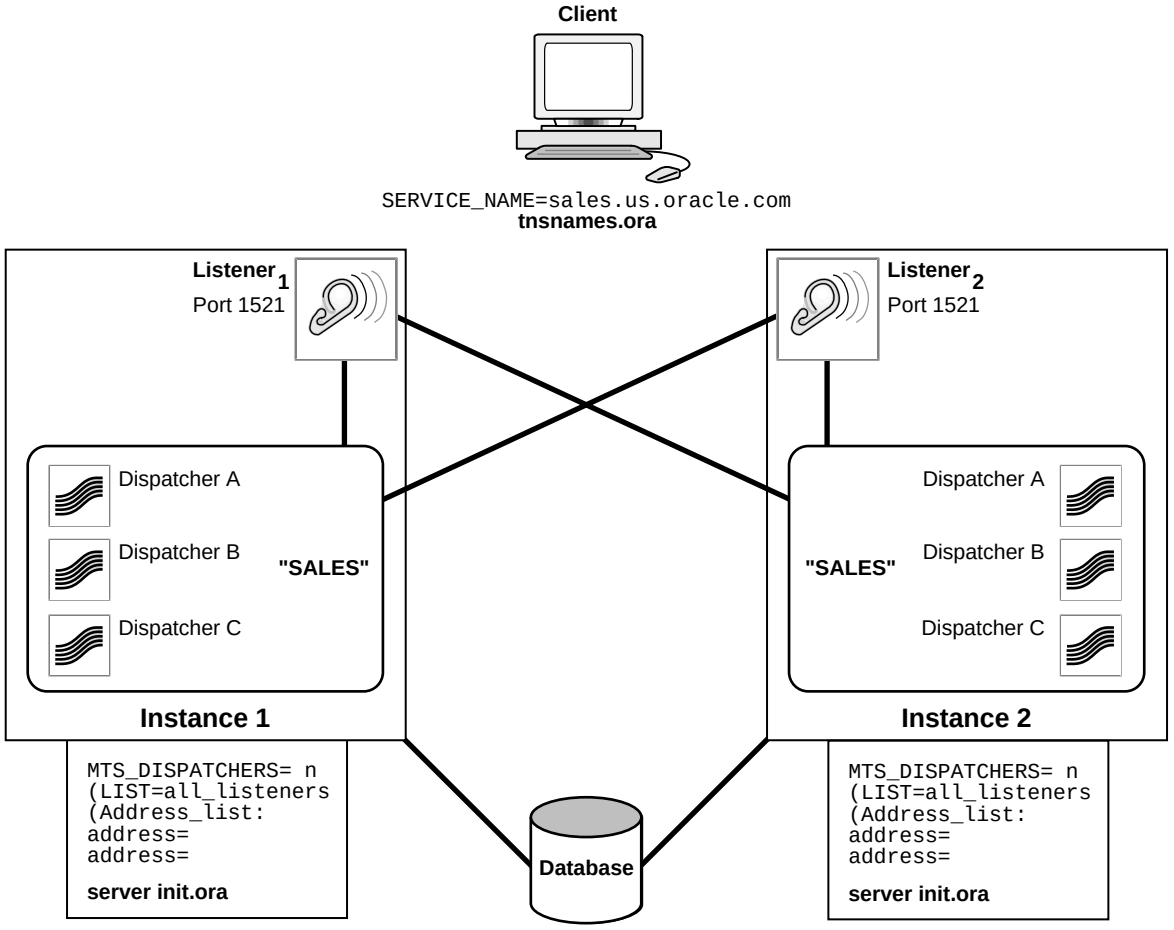
process of connect time failover. The processes spawned are then available to handle dedicated client server connections.

Instance registration refers to the recording of instance-specific information among all related listener processes. Most importantly, information about the load for a given instance is recorded among other listener processes within the same service.

How Clients Access Services

Clients connect to services by contacting the listeners as specified in the TNSNAMES.ORA file. The client passes the CONNECT_DATA from the TNSNAMES.ORA to the listener. This information tells the listener the service that the client wants to connect to. The listener then redirects the client to an appropriate handler or the listener creates a new dedicated server for the client.

Figure 18–4 Client-Service Connections



Configuring Client-to-service Connections

To configure client-to-service connections, or "client-to-instance" connections, make INIT.ORA entries as described in this section. This section also describes entries to TNSNAMES.ORA. For more information about configuring TNSNAMES.ORA, please refer to the *Net8 Administrator's Guide.*

This section explains how entries in these files influence the following OPS features:

- Database Instance Registration
- Connect Time Failover
- Client Load Balancing
- Connection Load Balancing

Database Instance Registration

Database instance registration is the process by which an instance "registers" with a listener. You can configure instances to register with local and remote listeners of the same service. To set up registration, configure the following INIT.ORA parameters:

```
SERVICE_NAMES = SALE.US.ORACLE.COM
LOCAL_LISTENER = protocol address
MTS_DISPATCHERS = (LIST = ... )
INSTANCE_NAME = instance name
```

For SERVICE_NAMES, enter a fully qualified service name as in this example. The value you enter for SERVICE_NAMES in INIT.ORA should be the same as the entry in TNSNAMES.ORA.

Note: There are alternate settings you can use instead of setting the SERVICE_NAMES parameter. For more information on these, please refer to the *Net8 Administrator's Guide.*

Connect Time Failover

Connect time failover refers to a client attempting to connect to a second listener when the attempt to connect to the first listener fails. You control how the client executes these connection attempts by the way you enter listener addresses in the address list within TNSNAMES.ORA.

To do this, enter listener addresses in TNSNAMES.ORA in the order in which you want the client to attempt to make client-to-service connections. Also set the TNSNAMES.ORA parameter FAILOVER to ON.

Connect time failover continues until the client successfully connects to a listener.

Note: Implementing connect time failover does not allow use of static service configuration parameters. Therefore, you cannot simultaneously implement Oracle Enterprise Manager (OEM) and connect time failover. OEM's Dynamic Discovery feature searches within a service for active connections using static configuration parameters.

See Also: For more information on Transparent Application Failover, please refer to *Oracle8i Tuning*.

Client Load Balancing

Client load balancing refers to the balancing of client connections among all listeners servicing a database. Enable this feature by setting the LOAD_BALANCE parameter in TNSNAMES.ORA to ON.

Client Load Balancing is not the same as "Parallel Query Load Balancing" which refers to dispersing server processes across instances to balance processing loads. Parallel query load balancing is described in more detail on page 18-33 under the heading "Parallel Execution Load Balancing".

Connection Load Balancing

Connection load balancing attempts to evenly distribute client connections among all available nodes, instances, and their dispatchers. The distribution of connections is based on each node's processing load and the number of active connections on each instance and on each dispatcher. Connection load balancing is automatically enabled when you configure the multi-threaded server.

Note: This feature is only available when you enable the Multi-threaded Server.

Parallel Execution Load Balancing

Parallel execution load balancing feature spreads server processes across instances to balance loads. This improves load balances for parallel execution and PDML operations on multiple instances.

Although you cannot tune this particular aspect of the automated degree of parallelism, you can adjust the database scheduler values to influence the load balancing algorithm of automated parallel query.

On MPP systems, Oracle first populates affinity nodes before populating non-affinity nodes. Generally, affinity nodes have loads that are about 10-15 percent higher than non-affinity nodes.

The load balancing feature uses your vendor-specific cluster manager software to communicate among instances.

See Also: For more information about degree of parallelism and its use, refer to *Oracle8i Tuning*.

Managed Standby and Standby Databases

You can protect OPS systems against disasters by using standby databases. To simplify the adminstration of standby databases, consider using the Managed Standby feature as described in the *Oracle8i Backup and Recovery Guide*.

19

Tuning to Optimize Performance

Last of the gods, Contention ends her tale.

Aeschylus, *Antigone*

This chapter provides an overview of Oracle Parallel Server (OPS) tuning issues. It covers the following topics:

- General Guidelines
- Contention
- Tuning for High Availability

> **See Also:** "Oracle Parallel Server Management" on page 18-2 for more information about using OPSM to administer multiple instances.

General Guidelines

This section covers the following topics:

- Overview
- Keep Statistics for All Instances
- Statistics to Keep
- Change One Parameter at a Time

Overview

With experience, you can anticipate most OPS application problems prior to rollout and testing of the application. Do this using the methods described in this chapter. In addition, a number of tunable parameters can enhance system performance. Tuning parameters can improve system performance, but they cannot overcome problems caused by poor analysis of potential Integrated Distributed Lock Manager (IDLM) lock contention.

Techniques you might use for single-instance applications are also valid in tuning OPS applications. It is still important, however, that you effectively tune the buffer cache, shared pool and all disk subsystems. OPS introduces additional parameters you must understand as well as OPS-specific statistics you must collect.

Note: Locks are mastered and remastered dynamically, so the instances do not need to be started in any particular order.

When collecting statistics to monitor OPS performance, the following general guidelines simplify and enhance the accuracy of your system debugging and monitoring efforts.

> **See Also:** For more information on mastering, please refer to "Lock Mastering" on page 8-9.

Keep Statistics for All Instances

It is important to monitor all instances in the same way but keep separate statistics for each instance. This is particularly true if the partitioning strategy results in a highly asymmetrical solution. Monitoring each instance reveals the most heavily loaded nodes and tests the performance of the system's partitioning.

> **See Also:** For more information on maintaining statistics, please refer to *Oracle8i Tuning*.

Statistics to Keep

The best statistics to monitor within the database are those kept within the SGA, for example, the "V$" and "X$" tables. It is best to "snapshot" these views over a period of time. In addition you should maintain an accurate history of operating system statistics to assist the monitoring and debugging processes. The most important of

these statistics are CPU usage, disk I/O, virtual memory use, network use and lock manager statistics.

Change One Parameter at a Time

In benchmarking or capacity planning exercises it is important that you effectively manage changes to the system setup. By documenting each change and effectively quantifying its effect, you can profile and understand the mechanics of the system and its applications. This is particularly important when debugging a system or determining whether more hardware resources are required. You must adopt a systematic approach for the measurement and tuning phases of a performance project. Although this approach may seem time consuming, it will save time and system resources in the long term.

Contention

This section covers the following topics:

- Detecting Lock Conversions
- Locating Lock Contention within Applications

Detecting Lock Conversions

To detect whether a large number of lock conversions is taking place, examine the "V$" tables that enable you to see locks the system is up- or and downgrading. The best views for initially determining whether lock contention problems exist are V$LOCK_ACTIVITY and V$SYSSTAT.

To determine the number of lock converts over a period of time, query the V$LOCK_ACTIVITY table. From this you can determine whether you have reached the maximum lock convert rate for the IDLM. If this is the case, you must repartition the application to remove the bottleneck. In this situation, adding more hardware resources such as CPUs, disk drives, and memory is unlikely to significantly improve system performance.

Note: The maximum lock convert rate depends on you platform's IPC mechanism implementation.

To determine whether there are too many lock conversions, calculate how often the transaction requires a lock conversion operation when a data block is accessed for either a read or a modification. To see this, query the V$SYSSTAT table.

In this way you can calculate a lock hit ratio that may be compared to the cache hit ratio. The value calculated is the number of times data block accesses occur that do not require lock converts, compared to the total number of data block accesses. The lock hit ratio is computed as:

$$\frac{consistent\_gets - global\_lock\_converts\_(async)}{consistent\_gets}$$

A SQL statement to compute this ratio is:

```
SELECT (b1.value - b2.value) / b1.value ops_ratio
  FROM V$SYSSTAT b1, V$SYSSTAT b2
WHERE b1.name = 'consistent gets'
  AND b2.name = 'global lock converts (async)';
```

If this ratio drops below 95%, you may not achieve optimal performance scaling as you add additional nodes.

Another indication of too many PCM lock conversions is the ping / write ratio, which is determined as follows:

$$ping\_write\_ratio = \frac{DBWR\_cross\_instance\_writes}{physical\_writes}$$

See Also: "Tuning PCM Locks" on page 15-14.

Locating Lock Contention within Applications

If an application shows performance problems and you determine that excessive lock conversion operations are the major problem, identify the transactions and SQL statements causing problems. When excessive lock contention occurs it is likely caused by one of three problem areas when setting up OPS. These areas are:

- Contention for a common resource

- Lack of locks
- Constraints

Excessive Lock Convert Rates: Contention for Common Resources

This section describes excessive lock conversion rates associated with contention for common resources.

In some cases within OPS, the system may not be performing as anticipated. This may be because the database setup or application design process overlooked some blocks that all instances must access for the entire time that the application runs. This forces the entire system to effectively "single thread" with respect to this resource.

This problem can also occur in single instance cases where all users require exclusive access to a single resource. In an inventory system, for example, all users may wish to modify a common stock level.

In OPS applications, the most common points for contention are associated with contention for a common set of database blocks. To determine whether this is happening, you can query an additional set of V$ tables. These are the VBH, VCACHE and V$PING tables.

These tables yield basically the same data, but V$CACHE and V$PING have been created as views joining additional data dictionary tables to make them easier to use. These tables and views examine the status of the current data blocks within an instance's SGA. They also enable you to construct queries to see how many times a block has been pinged between nodes and how many versions of the same data block exist within an SGA. You can use both of these features to determine whether excessive single threading upon a single database block is occurring.

Note: The GV$BH, GV$CACHE, and GV$PING views are also available, enabling you to query across all instances.

The most common areas of excessive block contention tend to be:

- **Free list contention by INSERT statements requiring more free space to insert into a table.** Often you can recognize this by querying V$PING and noticing that a single block has multiple copies in the SGA. If this is the second block in the file, free list contention is probably occurring. This problem may actually be solved using free list groups and multiple free lists. Free list contention may also occur on single-instance systems, especially SMP machines with a large

number of CPUs. This problem can be determined by querying the V$WAITSTAT table.

- **Segment header contention for transactions sharing the same space header management block.** This is likely to occur during parallel index creates when parallel query slaves allocate sorting space from the temporary tablespace. Each segment undergoing simultaneous space management in OPS requires approximately 9 distributed locks dedicated to coordinating space management activities.
- **Index contention by INSERT and DELETE statements that operate upon an indexed table.** By querying V$PING you can see that a number of data blocks within the first extent of the index have both multiple copies in the SGA and experience a high number of block pings. You cannot solve this problem by tuning. Instead, localize all access to this index to a single instance. This includes both read and write access.

  This involves routing transactions altering this table to a single instance and running the system asymmetrically. If you cannot do this, consider partitioning the table and using a data dependent routing strategy.

Excessive Lock Convert Rates through Lack of Locks

In tables with random access for SELECT, UPDATE and DELETE statements, each node needs to perform a number of PCM lock up- and downgrades. If these lock conversion operations require disk I/O, they will be particularly expensive and adversely affect performance.

If, however, many of the lock converts can be satisfied by just converting the lock without a disk I/O, a performance improvement can be made. This is often referred to as an I/O-less ping, or a ping not requiring I/O. The reason the lock convert can be achieved without I/O is that the database is able to age data blocks out of the SGA using DBWR as it would with a single instance. This is only likely when the table is very large compared to the size of the SGA. Small tables are likely to require disk I/O, since they are unlikely to age out of the SGA.

With small tables where random access occurs you can still achieve performance improvements by using aggressive checkpointing. To do this, reduce the number of rows stored in a data block by increasing the table PCTFREE value and by reducing the number of data blocks managed by a PCM lock. The process of adjusting the number of rows managed per PCM lock can be performed until lock converts are minimized or the hardware configuration runs out of PCM locks.

The number of PCM locks managed by the IDLM is not an infinite resource. Each lock requires memory on each OPS node, and this resource may be quickly be

exhausted. Within an OPS environment, the addition of more PCM locks lengthens the time taken to restart or recover an OPS instance. In environments where high availability is required, the time taken to restart or recover an instance may determine the maximum number of PCM locks you can practically allocate.

Excessive Lock Convert Rates Due to Constraints

In certain situations, excessive lock conversion rates cannot be reduced due to certain constraints. In large tables, clusters, or indexes many gigabytes in size, it becomes impossible to allocate enough PCM locks to prevent high lock convert rates even if these are all false pings. This is mainly due to the physical limitations of allocating enough locks. In this situation, a single PCM lock may effectively manage more than a thousand data blocks.

Where random access is taking place, lock converts are performed even if there is no contention for the same data block. In this situation, tuning the number of locks is unlikely to enhance performance since the number of locks required is far in excess of what can actually be created by the IDLM.

In such cases you must either restrict access to these database objects or else develop a partitioned solution.

Tuning for High Availability

Failure of an Oracle instance on one OPS node may be caused by problems that may or may not require rebooting the failed node. If the node fails and requires a reboot or restart, the recovery process on remaining nodes will take longer. Assuming a full recovery is required, the recovery process will be performed in three discreet phases:

- Detection of Error
- Recovery and Re-mastering of IDLM Locks
- Recovery of Failed Instance

Detection of Error

The first phase of recovery is to detect that either a node or an OPS instance has failed. Complete node failure or failure of an Oracle instance is detected through the operating system node management facility.

Recovery and Re-mastering of IDLM Locks

If a complete node failure has occurred, the remaining nodes must remaster locks held by the failed node. On non-failed instances, all database processing stops until recovery has completed. To speed IDLM processing it is important to have the minimum number of PCM locks. This will eventually be reflected in a trade-off between database performance and availability requirements.

Recovery of Failed Instance

Once the IDLM has recovered all lock information, one of the remaining nodes can get an exclusive lock on the failed instance's IDLM instance lock. This node enables the failed instance to provide roll forward / roll backward recovery of the failed instance's redo logs. This is performed by the SMON background process. The time needed to perform this process depends upon the number of redo logs to be recovered. This is a function of how often the system was checkpointed at runtime. Again, this is a trade-off between system runtime performance, which favors a minimum of checkpoints, and system availability requirements.

See Also: "Phases of Oracle Instance Recovery" on page 22-5.

20

Cache Fusion and Inter-instance Performance

This chapter explains how Cache Fusion resolves reader / writer conflicts in Oracle Parallel Server. It describes Cache Fusion and its benefits in general terms that apply to most types of systems and applications. The chapter also describes OPS- and Cache Fusion-related statistics and provides many procedures that explain how to use these statistics to monitor and tune performance.

The topics in this chapter are:

- The Role of Cache Fusion in Resolving Cache Coherency Conflicts
- How Cache Fusion Produces Consistent Read Blocks
- Improved Scalability with Cache Fusion
- The Interconnect and Interconnect Protocols for OPS
- Performance Expectations
- Monitoring Cache Fusion and Inter-instance Performance

Note: For an overview of Cache Fusion processing, please refer to "Cache Fusion Processing and the Block Server Process" on page 5-6.

The Role of Cache Fusion in Resolving Cache Coherency Conflicts

Inter-instance contention for data blocks and the resulting cache coherency issues are the main performance problems of OPS. In most cases, proper partitioning resolves most contention problems.

In reality, however, most packaged applications are not effectively partitioned, or are partitioned only to a limited extent. Such applications experience 3 types of inter-instance contention:

- Reader / Reader
- Reader / Writer
- Writer / Writer

Reader / writer contention occurs when one instance needs to read a data block in consistent mode and the correct version of the block does not exist in the instance's cache. OPS 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.

In the case of inserts into tables, for example, writer / writer conflicts are partially addressed by free list groups. In other cases, however, the only alternative is to address writer / writer cache coherency issues by isolating hot blocks using locking, by implementing deferred pinging, or by application partitioning. Reader / writer conflicts, on the other hand, are more prevalent and easier to resolve.

Reader / writer contention is the most common type of contention in OLTP and hybrid applications. The ability to combine DSS and OLTP processing in a typical application depends on OPS' efficiency in resolving such conflicts.

For the "reader" part of reader / writer conflicts there are two subcategories: the contention caused by current readers and contention caused by consistent read readers. Of these two, consistent read readers are typically more prevalent and Cache Fusion directly addresses these.

How Cache Fusion Produces Consistent Read Blocks

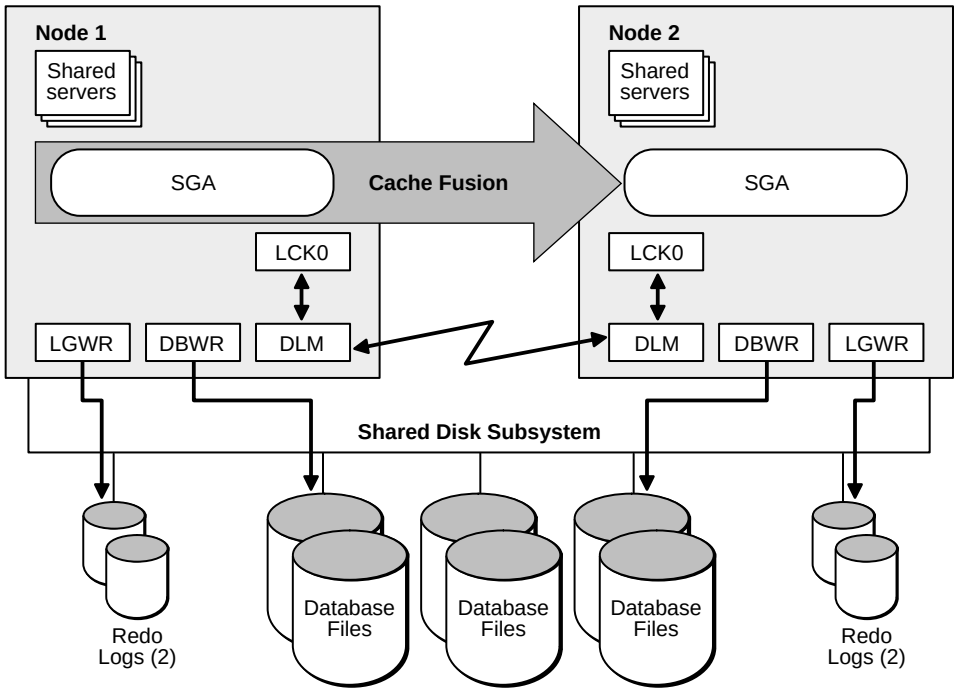
If a data block requested by one instance is in the memory cache of a remote instance, Cache Fusion resolves the conflict using remote memory access, not disk access. The requesting instance sends a request for a consistent-read copy of the block to the holding instance. The BSP (Block Server Process) on the holding instance transmits the consistent-read image of the requested block directly from

the holding instance's buffer cache to the requesting instance's buffer cache across a high speed interconnect.

As Figure 21–1 illustrates, Cache Fusion enables the buffer cache of one node to ship data blocks directly to the buffer cache of another node by way of low latency, high bandwidth interconnects. This reduces the need for expensive disk I/O in parallel cache management.

Cache Fusion also leverages new interconnect technologies for low latency, user-space based, interprocessor communication. This drastically lowers CPU usage by reducing operating system context switches for inter-node messages.

Figure 20–1 Cache Fusion Ships Blocks from Cache to Cache Across the Interconnect



Oracle manages write/write contention using conventional disk-based PCM (Parallel Cache Management). A later version of Oracle will use Cache Fusion to provide faster writer/writer contention resolution.

Note: Cache Fusion is always enabled.

See Also: For more information on partitioning, please refer to *Oracle8i Concepts*

Partitioning Data to Improve Write/write Conflict Resolution

Cache Fusion only solves part of the block conflict resolution issue by providing improved scalability for applications that experience high levels of reader / writer contention. For applications with high writer / writer contention levels, you also need to accurately partition your application's tables to reduce the potential for writer / writer conflicts.

See Also: For more information on partitioning, please refer to *Oracle8i Tuning* and *Oracle8i Concepts.*

Improved Scalability with Cache Fusion

Cache Fusion improves application transaction throughput and scalability by providing:

- Greatly reduced operating system context switches, and hence reduced CPU utilization, during reader / writer cache coherency conflicts
- Further reduced CPU utilization for user-mode IPC platforms
- Reduced I / O for block pinging and reduced X-to-S lock conversions
- Consistent-read block transfers by way of high speed interconnects

Applications demonstrating high reader / writer conflict rates under disk-based PCM benefit the most from Cache Fusion.

Note: If 20% or more of your application's inter-instance activity was due to reader / writer contention, you will likely see significant performance gains from Cache Fusion.

Packaged applications also scale more effectively as a result of Cache Fusion. Applications in which OLTP and reporting functions execute on separate nodes may also benefit from Cache Fusion. Reporting functions that access data from

tables modified by OLTP functions receive their versions of data blocks by way of high-speed interconnects. This reduces the 'pinging' of data blocks to disk.

Performance gains are derived primarily from reduced X-to-S lock conversions and the corresponding reduction in disk I/O for X-to-S lock conversions.

Note: All applications achieve some performance gains from Cache Fusion. The degree of improvement depends upon the operating system, the application workload, and the overall system configuration.

The direct mapping of data buffers across instances during inter-processor communication also avoids having to copy memory from one address space to another. This shortened execution path reduces CPU requirements and increases the communications bandwidth as explained under the following headings.

Reduced Context Switches and CPU Utilization

Cache Fusion dramatically reduces operating system context switches. This results in reduced CPU utilization and frees CPU cycles for applications processing.

Reduced CPU Utilization with User-mode IPCs

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

User-mode IPCs reduce CPU utilization because user processes can communicate without using the operating system kernel. In other words, there is no need to switch from user execution mode to kernel execution mode.

Reduced I/O for Block Pinging and Reduced X-to-S Lock Conversions

Cache Fusion reduces expensive lock operations and disk I/O for data and undo segment blocks by transmitting consistent-read blocks directly from one instance's buffer cache to another. This can reduce the latency required to resolve reader/writer conflicts by as much as 90%.

Disk-based PCM may require as much as 80ms (milliseconds) to resolve reader/writer conflicts. This involves disk I/O for the requested block as well as I/O to write rollback segment blocks to disk.

Cache Fusion resolves reader/writer conflicts with approximately 1/10th the processing effort required by disk-based PCM using little or no disk I/O. To do this, Cache Fusion only incurs overhead for processing the consistent-read request and for constructing a consistent-read copy of the requested block in memory and transferring it to the requesting instance. On some platforms this can take less than 1ms.

Consistent-read Block Transfers by way of High Speed Interconnects

Because Cache Fusion exploits high speed IPCs, OPS benefits from the performance gains of the latest technologies for low latency communication across cluster interconnects. Further performance gains can be expected with even more efficient protocols, such as VIA and user-mode IPCs.

The Interconnect and Interconnect Protocols for OPS

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

Influencing Interconnect Processing

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

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

Supported Interconnect Software

Interconnects supporting OPS and Cache Fusion use one of these protocols:

- TCP/IP (Transmission Control Protocol/Interconnect Protocol)
- UDP (User Datagram Protocol)
- VIA (Virtual Interconnect Architecture)
- Other proprietary protocols that are hardware vendor-specific

OPS can use any interconnect product that supports these protocols. The interconnect product must also be certified for OPS hardware cluster platforms.

Performance Expectations

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

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

As mentioned earlier, if your application did not demonstrate a significant amount of consistent-read contention prior to Cache Fusion, your performance with Cache Fusion will likely remain unchanged. However, if your application experienced numerous lock conversions and heavy disk I/O as a result of consistent-read conflicts, your performance with Cache Fusion should improve dramatically.

As an example, query statistics in V$SYSSTAT and you should observe that your system's processing with Cache Fusion has fewer X-to-S lock converts. The fewer X-to-S lock conversions your application generates, the less disk I/O your system requires. The following section, "Monitoring Cache Fusion and Inter-instance Performance" on page 21-9, describes how to evaluate Cache Fusion performance in more detail.

> **See Also:** For more information on lock types, please refer to Chapter 7, "Overview of Locking Mechanisms".

Cache Fusion Block Request Latencies

Block request latencies in Cache Fusion can vary according to the protocol in use. With TCP / UDP, the latency can range from 5 to 50ms. The performance statistics achieved with TCP / UDP protocols, however, vary from system to system.

With VIA, which is only available on NT, the latency can be less than 1ms. Performance statistics for the VIA protocol do not vary significantly because VIA uses fewer context switches. Fewer context switches mean reduced overhead.

Monitoring Cache Fusion and Inter-instance Performance

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

- Goals of Monitoring Cache Fusion and OPS Performance
- Statistics for Monitoring OPS and Cache Fusion
- Using V$SYSTEM_EVENTS to Identify Performance Problems

Goals of Monitoring Cache Fusion and OPS Performance

The main goal of monitoring Cache Fusion and OPS performance is to examine the latency and fine-tune your system's processing by observing trends over time. Do this by analyzing the performance statistics from several views as described in the following sections. Use these monitoring procedures on an on-going basis to observe processing trends and to maintain processing at optimal levels.

Latency Statistics in OPS

The procedures in the following sections describe performance issues in terms of latency. The procedures also describe how to analyze other performance-related issues such as Integrated Distributed Lock Manager (IDLM) resource use and the status of general system events.

The Role of Latency in OPS Processing

Latency is the most important aspect of OPS and Cache Fusion performance. Latency is the amount of time required to complete a request for a consistent-read block. Latency is influenced by the type of requests and responses involved in consistent-read operations. Each type of request may have a different outcome as described in the following:

- When an instance needs a consistent-read version of a data block, it sends a lock request to the IDLM to determine which instance is currently modifying the data block. This request is typically for a shared lock. If a data block is currently being modified by another instance, the IDLM forwards the consistent-read request to the owner of the X lock for that block. If the block is not being modified by an instance, the IDLM grants an S lock to the requesting instance and the requesting instance reads the block from disk.
- The BSP on the instance holding the lock for the requested block uses messaging to prepare and transfer a consistent-read version of the block to the

requesting node. BSP first looks for the version of the requested block in its cache. If the correct version is not there, the holding instance builds the version using rollback segment information from the local cache. This avoids disk I/O.

- The holding instance may respond to a requesting instance with a message granting the requestor permission to read the block from disk when the requested block is no longer in the holding instance's cache. In other words, while most consistent-read requests are filled without disk I/O, some transactions may require the requestor to read the block from disk.

Statistics for Monitoring OPS and Cache Fusion

Oracle collects Cache Fusion-related performance statistics from the buffer cache and IDLM layers. Oracle also collects general OPS statistics for lock requests and lock waits. You can use several views to examine these statistics.

Maintaining an adequate history of system performance helps you more easily identify trends as these statistics change. This is especially important for identifying increasing latencies and adverse workload changes.

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

- Analyzing Global Cache and Cache Fusion Statistics
- Analyzing Global Lock Statistics
- Analyzing IDLM Resource, Lock, Message, and Memory Resource Statistics
- Analyzing OPS I/O Statistics
- Analyzing Lock Conversions by Type
- Analyzing Latch, OPS, and IDLM-related Statistics

In many cases, resolving performance issues requires that you first identify a problem using the specific procedures in each statistics group. You then use the V$SYSTEM_EVENT view to pinpoint the cause as described on page 21-33 under the heading, "Events in V$SYSTEM_EVENTS Specifically Related to OPS".

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

Note: You must also run CATPARR.SQL to create OPS-related views and tables for storing and viewing statistics as described under the next heading.

Creating OPS Data Dictionary Views with CATPARR.SQL

The SQL script CATPARR.SQL creates parallel server data dictionary views. To run this script, you must have SYSDBA privileges.

CATALOG.SQL creates the standard V$ dynamic views, as described in the *Oracle8i Reference*, as well as:

- GV$CACHE
- GV$PING
- GV$CLASS_PING
- GV$FILE_PING

You can rerun CATPARR.SQL if you want the EXT_TO_OBJ table to contain the latest information after you add extents. If you drop objects without rerunning CATPARR.SQL, EXT_TO_OBJ may display misleading information.

> **See Also:** *Oracle8i Reference* for more information on dynamic views and monitoring your database.

Global Dynamic Performance Views

Tuning and performance information for the Oracle database is stored in a set of dynamic performance tables known as the "V$ fixed views". Each active instance has its own set of fixed views. In OPS, you can query a global dynamic performance (GV$) view to retrieve the V$ view information from all qualified instances. A global fixed view is available for all of the existing dynamic performance views except for V$ROLLNAME, V$CACHE_LOCK, V$LOCK_ACTIVITY, and V$LOCKS_WITH_COLLISIONS.

The global view contains all the columns from the local view, with an additional column, INST_ID (datatype INTEGER). This column displays the instance number from which the associated V$ information was obtained. You can use the INST_ID column as a filter to retrieve V$ information from a subset of available instances. For example, the query:

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

Retrieves information from the V$ views on instances 2 and 5.

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

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

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

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

For global views, it may be acceptable to continue running the query in parallel and return the data from the instances that could allocate servers for the query. If it is acceptable to retrieve results only from instances where server allocation succeeded, set the value to TRUE.

See Also: "Specifying Instance Groups" on page 18-23 and *Oracle8i Reference* for restrictions on GV$ views and complete descriptions of all related parameters and V$ dynamic performance views.

Analyzing Global Cache and Cache Fusion Statistics

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

Requests for global locks on data blocks originate in the buffer cache of the requesting instance. Before a request enters the IDLM, Oracle allocates data structures in the SGA to track the state of the request. These structures are called "lock elements".

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

Procedures for Monitoring Global Cache Statistics

Complete the following steps to analyze global cache statistics.

1. Use this syntax to query V$SYSSTAT:

```
SELECT * FROM V$SYSSTAT WHERE NAME LIKE 'global cache';
```

Oracle responds with output similar to:

```
NAME                                                                  VALUE
---------------------------------------------------------------- ----------
global cache gets                                                      12480
global cache get time                                                    996
global cache converts                                                     21
global cache convert time                                                 48
global cache cr blocks received                                            1
global cache cr block receive time                                         1
global cache cr read from disk                                             0
global cache freelist waits                                                0
global cache defers                                                        0
global cache convert timeouts                                              0
global cache cr timeouts                                                   0
global cache fairness down converts                                        0
```

Use your V$SYSSTAT output to perform the calculations and analyses described in the remaining procedures for this statistics group.

Procedures 2 and 3 use the following Cache Fusion statistics from V$SYSSTAT:

- global cache cr blocks received
- global cache cr blocks receive time
- global cache cr read from disk

- global cache cr timeouts
- global cache fairness down converts

2. Calculate the latency for Cache Fusion requests using this formula:

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

The result, which typically varies from 5 to 40ms depending on your system configuration and volume, is the average latency of a consistent-read request round trip from requesting instance, to holding instance, and back to the requesting instance. If your CPU has limited idle time and your system typically processes long-running queries, the latency may be higher. However, it is possible to have an average latency of less than 1ms if your interconnect protocol is user-mode IPC.

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

3. Calculate the total number of consistent-read requests using this formula:

global cache cr block received + global cache cr blocks read from disk

A high proportion of read permissions from disk indicates that blocks are rapidly aging out of the buffer cache of the holding instance. To resolve this, increase the size of your buffer cache.

Procedures 4 and 5 require that you take snapshots of your statistics, for example, by using UTLBSTAT and UTLESTAT.

> **See Also:** For more information on UTLBSTAT and UTLESTAT, please refer to *Oracle8i Tuning.*

Procedure 4 uses the following global cache statistics from V$SYSSTAT:

- global cache convert time
- global cache converts
- global cache get time
- global cache gets

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

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

If the average convert or get time is high, there is excessive contention. Another cause may be that latencies for lock operations are high due to overall system workload or system problems. A reasonable value for a cache gets is 20-30ms while converts should take 10-20ms on average.

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

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

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

5. Analyze lock convert timeouts by examining the value for 'global cache convert timeouts'. If your V$SYSSTAT output shows a value other than zero (0) for this statistic, check your system for congestion or high contention. In general, convert timeouts should not occur; their existence indicates serious performance problems.
6. Analyze the global cache consistent-read timeouts by examining the value for this statistic in your V$SYSSTAT output. Oracle increments this statistic after the system waits too long for the completion of a consistent-read request. If this statistic shows a value other than zero (0), too much time has elapsed after the

initiation of a consistent-read request and a timeout has occurred. If this happens, you will also usually find that the average time for consistent-read request completions has increased. If you have timeouts and the latency is high, your system may have an excessive workload or there may be excessive contention for data blocks.

Analyzing Global Lock Statistics

Global lock statistics provide counts and timings for both PCM and non-PCM lock activity. Oracle collects global lock statistics from the IDLM API layer. All Oracle clients to the IDLM, of which the buffer cache is only one, make their requests to the IDLM through this layer. Thus, global lock statistics include lock requests originating from all layers of the kernel, while global lock statistics relate to buffer cache OPS activity.

Use procedures in this section to monitor data from V$SYSSTAT to derive averages, latencies, and counts. This establishes a rough indicator of the OPS workload generated by an instance.

Procedures for Analyzing Global Lock Statistics

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

1. Use this syntax to query V$SYSSTAT:

```
SELECT * FROM V$SYSSTAT WHERE NAME LIKE 'global lock';
```

Oracle responds with output similar to:

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

Use your V$SYSSTAT output to perform the calculations and analyses described in the remaining procedures in this statistics group.

2. Calculate the average global lock gets using this formula:

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

If the result is more than 20 or 30ms, query the TIME_WAITED column in V$SYSTEM_EVENTS using the DESCEND keyword to identify which lock events are waited for most frequently using this query:

```
SELECT EVENT_TIME_WAITED, AVERAGE_WAIT
FROM V$SYSTEM_EVENTS
ORDER BY TIME_WAITED DESCEND;
```

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

The elapsed time for a get thus includes the allocation and initialization of new locks. If the average lock get or average lock convert times are excessive, your system may be experiencing timeouts.

If the global lock convert times or global lock get times are high, refer to statistics in the V$SYSTEM_EVENTS view to identify events with a high value for TIME_WAITED statistics.

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

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

If the result is more than 20ms, query the TIME_WAITED column in V$SYSTEM_EVENTS using the DESCEND keyword to identify the event causing the delay.

4. Analyze the V$LIBRARYCACHE and V$ROWCACHE views to observe IDLM activity on non-PCM resources. These views have IDLM-specific columns that identify IDLM resource use. Do this if you have frequent and extended waits for library cache pins, enqueues, or DFS lock handles.

Analyzing IDLM Resource, Lock, Message, and Memory Resource Statistics

Oracle collects IDLM resource, lock, and message statistics at the IDLM level. Use these statistics to monitor IDLM latency and workloads. These statistics appear in the V$DLM_CONVERT_LOCAL and V$DLM_CONVERT_REMOTE views.

These views record average convert times, count information, and timed statistics for each type of lock request. V$DLM_CONVERT_LOCAL shows statistics for local lock operations. V$DLM_CONVERT_REMOTE shows values for remote conversions. The average convert times in these views are in 100ths of a second.

Note: Count information in these views is cumulative for the life of an instance.

How IDLM Workloads Affect Performance

The IDLM workload is an important aspect of OPS and Cache Fusion performance because each consistent-read request results in a lock request. High IDLM workloads as a result of heavy request rates can adversely affect performance.

The IDLM performs local lock operations entirely within the local node, or in other words, without sending messages. Remote lock operations require sending messages to and waiting for responses from other nodes. Most down-converts, however, are local operations for the IDLM.

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

Procedures for Analyzing IDLM Resource and Lock Statistics

Use the following procedures to obtain and analyze statistics from V$DLM_CONVERT_LOCAL and V$DLM_CONVERT_REMOTE for DLM resource processing.

You must enable event 29700 to populate the V$DLM_CONVERT_LOCAL and V$DLM_CONVERT_REMOTE views. Do this by entering this syntax:

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

1. Use this syntax to query V$DLM_CONVERT_LOCAL:

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

Oracle responds with output similar to:

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

2. Use this syntax to query V$DLM_CONVERT_REMOTE:

```
SELECT * FROM V$DLM_CONVERT_REMOTE;
```

Oracle responds with output identical in format to the output for V$DLM_CONVERT_LOCAL.

Use your output from V$DLM_CONVERT_LOCAL and V$DLM_CONVERT_REMOTE to perform the calculation described in the following procedure.

3. Calculate the ratio of local-to-remote lock operations using this query:

```
SELECT r.CONVERT_TYPE,
 r.AVERAGE_CONVERT_TIME,
 l.AVERAGE_CONVERT_TIME,
 r.CONVERT_COUNT,
 l.CONVERT_COUNT,
```

```
FROM V$DLM_CONVERT_LOCAL l, V$DLM_CONVERT_REMOTE r
GROUP BY r.CONVERT_TYPE;
```

4. It is useful to maintain a history of workloads and latencies for lock converts. Changes in request rates for lock operations without increases in average latencies usually results from changing application workload patterns.

   Deterioration of both request rates and latencies usually indicates an increased rate of lock conflicts or an increased workload due to message latencies, system problems, or timeouts. If the LMD process shows high CPU consumption, or consumption that is greater than 20% of the CPU, while overall system resource consumption is normal, consider binding the LMD process to a specific processor if the system has more than one CPU.

   If latencies increase, also examine CPU data and other operating system statistics that you can obtain using utilities such as "sar," "vmstat" and "netstat".

IDLM Message Statistics

The IDLM sends messages either directly or by using flow control. For both methods, the IDLM attaches markers known as "tickets" to each message. The allotment of tickets for each IDLM is limited. However, the IDLM can re-use tickets indefinitely.

IDLMs send messages directly until no more tickets are available. When an IDLM runs out of tickets, messages must wait in a flow control queue until outstanding messages finish processing and more tickets are available. Flow-controlled messaging is managed by the LMD process.

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

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

- DLM messages sent directly
- DLM messages flow controlled
- DLM messages received
- DLM total incoming message queue length

Procedures for Analyzing IDLM Message Statistics

Use the following procedures to obtain and analyze message statistics in V$DLM_MISC.

1. Use this syntax to query V$DLM_MISC:

```
SELECT NAME, VALUE FROM V$DLM_MISC;
```

Oracle responds with output similar to:

```
STATISTIC# NAME
---------- -----------------------------------VALUE-----
         0 dlm messages sent directly           29520

         1 dlm messages flow controlled          1851

         2 dlm messages received                29668

         3 dlm total incoming msg queue length    297

4 rows selected.
```

> **Note:** Oracle support may request information from your V$DLM_MISC output for debugging purposes.

Use your output from V$DLM_MISC to perform the following procedure.

2. Calculate the average receive queue length between two snapshots using this equation:

$$\frac{\textit{total incoming message queue length}}{\textit{messages received}}$$

Oracle increments the value for 'total incoming message queue length' whenever a new request enters the LMD process' message queue. When messages leave the LMD queue to begin processing, Oracle increments the value for 'messages received'.

The size of the queue may increase if a large number of requests simultaneously arrives at the LMD. This can occur when the volume of locking activity is high or when the LMD processes a large quantity of consistent-read requests. Typically, the average receive queue length is less than 10.

Analyzing OPS I/O Statistics

In addition to the global cache and global lock statistics that were previously discussed, you can also use statistics in V$SYSSTAT to measure the I/O workload related to global conflict resolution. There are three important sets of statistics in V$SYSSTAT for this purpose:

- DBWR forced writes
- Remote instance undo header writes
- Remote instance undo block writes

DBWR forced writes occur when Oracle resolves inter-instance data block contention by writing the requested block to disk before the requesting node can use it.

If a consistent-read request requires information from another instance's cache to roll back a block and make it read consistent, Oracle must also write rollback segment headers and rollback segment blocks to disk. One instance must write the undo blocks and undo headers to disk while another instance reads them.

Cache Fusion minimizes the disk I/O for consistent-reads. This can lead to a substantial reduction in physical writes performed by each instance. Before Cache Fusion, a consistent-read involving data from a remote instance required up to 3 writes, 3 reads, a rollback segment header, an undo segment block, and multiple lock converts for one requested block.

Procedures for Analyzing OPS I/O Statistics

Use the following procedures to obtain and analyze message statistics in V$SYSSTAT.

1. Use this syntax to query V$SYSSTAT:

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ('DBWR forced writes',
'remote instance undo block writes',
'remote instance undo header writes',
'physical writes');.
```

Oracle responds with output similar to:

```
NAME                                                          VALUE
------------------------------------------------------------ ----------
physical writes                                              41802
DBWR forced writes                                            5403
remote instance undo block writes                                0
remote instance undo header writes                               2
4 rows selected.
```

Use your V$SYSSTAT output to perform the following calculations.

2. Calculate the ratio of OPS-related I/O to overall physical I/O using this equation:

$$\frac{DBWR\ forced\ writes}{physical\ writes}$$

You should see a noticeable decrease in this ratio between this calculation and pre-Cache Fusion statistics.

3. Calculate how many writes to rollback segments occur when a remote instance needs to read from rollback segments that are in use by a local instance using this equation:

$$\frac{\textit{(remote instance undo header writes + remote instance undo block writes)}}{\textit{DBWR forced writes}}$$

The ratio shows how much disk I/O is related to writes to rollback segments. With Cache Fusion, this ratio should be very low.

Analyzing Lock Conversions by Type

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

- Using V$LOCK_ACTIVITY to Analyze Lock Conversions
- Using V$CLASS_PING to Identify Pinging by Block Class
- Using V$PING to Identify Hot Objects

> **See Also:** For more information about lock conversions, please refer to Chapter 8, "Integrated Distributed Lock Manager".

Using V$LOCK_ACTIVITY to Analyze Lock Conversions

V$LOCK_ACTIVITY summarizes how many lock up- and down-converts have occurred during an instance's lifetime. X-to-N down-converts denote the number of times a lock was down-converted because another instance wanted to modify a resource.

The other major type of down-convert is X-to-S. This type of down-convert occurs when an instance reads a resource that was last modified by a local instance. Both types of lock conversions involve I/O. However, Cache Fusion should reduce X-to-S down-converts because they are not needed for buffer locks.

Using V$CLASS_PING to Identify Pinging by Block Class

V$CLASS_PING summarizes lock conversion activity by showing whether disk I/O is occurring on the following classes of blocks:

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

All X_2_NULL_FORCED_WRITE and X_2_S_FORCED_WRITE conversions involve write I/O. In other words, values in the columns for each block class provide an indicator of the cause of the disk I/O.

Using V$PING to Identify Hot Objects

V$PING helps identify "hot" blocks and "hot" objects. The sum of the columns FORCED_READS and FORCED_WRITES indicates the actual pinging activity on a particular block or object.

> **See Also:** For more information about V$PING, please refer to "Querying the V$PING View to Detect Pinging" on page 20-10.

All three views provide different levels of detail. If you suspect that pinging or OPS itself is the cause of a performance problem, monitor V$LOCK_ACTIVITY to generate an overall OPS workload profile. Use information from V$LOCK_ACTIVITY to record the rate at which lock conversions occur.

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

Analyzing Latch, OPS, and IDLM-related Statistics

Latches are low level locking mechanisms that protect SGA data structures. Excessive contention for latches degrades performance.

Use V$DLM_LATCH and V$LATCH_MISSES to monitor latch contention within the IDLM. These views show information about a particular latch, its statistics, and the location in the code from where the latch is acquired.

Procedures for Analyzing Latch, OPS, and IDLM-related Statistics

Use the following procedures to analyze latch, OPS, and IDLM-related statistics.

1. Query V$LATCH using this syntax:

```
SELECT * FROM V$LATCH;
```

Oracle responds with output similar to:

```
cache buffer handle             46184          1        1          0          0
cache buffers chained        84139946     296547     .996      29899       .101
cache buffers lru             4760378      11718     .998        227       .019
channel handle pool                 1          0        1          0          0
channel operations                  1          0        1          0          0
dlm ast latch                  542776        494     .999       1658      3.356
dlm cr bast queue               37194          1        1          0          0
dlm deadlock list               32839          0        1          0          0
dlm domain lock la                  1          0        1          0          0
dlm domain lock ta              49164          1        1          0          0
dlm group lock lat                  1          0        1          0          0
dlm group lock tab              25239          1        1          0          0
dlm lock table fre             325306        270     .999        327      1.211
dlm process hash l               6346          0        1          0          0
dlm process table                   2          0        1          0          0
dlm recovery domai               2014          0        1          0          0
dlm resource hash              683031       1709     .997      41342     24.191
dlm resource scan                 188          0        1          0          0
dlm resource table             182093         70        1          2       .029
dlm shared communication       190766        211     .999        313      1.483
dlm timeout list               113294         40        0          3       .075
dml lock allocation               261          0        1          0          0
22 rows selected.
```

> **Note:** The content of the five columns in this output example from left to right are: gets, hits, misses, sleeps, and the sleeps-to-misses ratio.

2. If the output from the previous procedure reveals a high ratio of sleeps-to-misses, attempt to determine where the sleeps occur. To do this, execute this query on the V$LATCH_MISSES view:

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

Oracle responds with output similar to:

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

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

3. Calculate the ratio of gets to misses using your V$LATCH output from the first procedure in this group on page 21-30 in this formula:

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

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

4. Analyze the ratio of sleeps to misses using your V$LATCH_MISSES output from procedure 2 on page 21-31. This ratio determines how often a process sleeps when it cannot immediately get a latch but wants to wait for the latch.

   A ratio of 2 means that for each miss, a process attempts to get a latch twice before acquiring it. A high number of sleeps-to-misses usually indicates process scheduling delays or high operating system workloads. It can also indicate internal inefficiencies or high concurrency on one resource. For example, when many locks are opened simultaneously on the same resource, then processes might have to wait for a resource latch.

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

The following section describes how to use V$SYSTEM_EVENTS in more detail.

Using V$SYSTEM_EVENTS to Identify Performance Problems

Data about Cache Fusion and OPS events appears in the V$SYSTEM_EVENT view. To identify events for which processes have waited the longest, query V$SYSTEM_EVENT on the TIME_WAITED column using the DESCENDING keyword. The TIME_WAITED column shows the total wait time for each system event listed. For an example of how to query V$SYSTEM_EVENTS, refer to Step 2 on page 21-17.

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

Events in V$SYSTEM_EVENTS Specifically Related to OPS

The following events appearing in V$SYSTEM_EVENT output represent waits for OPS events:

- Global cache cr request
- Library cache pin
- Buffer busy due to global cache
- Global cache lock busy
- Global cache lock open x
- Global cache lock open s
- Global cache lock null to x
- Global cache lock s to x
- Global cache lock null to s

Events Related to Non-PCM Resources

You can monitor other events in addition to those listed under the previous heading because performance problems may be related to OPS. These events are:

- Row cache locks
- Enqueues
- Library cache pins

- DFS lock handle

General Observations

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

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

If a lot of wait time is used by waits for non-buffer cache resources as indicated by statistics in the rows 'row cache lock', 'enqueues', and 'library cache pin', monitor the V$ROWCACHE and V$LIBRARYCACHE views for OPS-related issues. Specifically, observe values in the IDLM columns of each of these views.

Common OPS problems arise from poorly managed space parameters or sequences that are not cached. In such cases, processes wait for row cache locks and enqueues and V$ROWCACHE will show a high number of conflicts for certain dictionary caches.

21

Backing Up the Database

Those behind cried "Forward!"
And those before cried "Back!"

Thomas Babington, Lord Macaulay: *On Frederic The Great*

To protect your data, archive the online redo log files and periodically back up the datafiles. Also back up the control file for your database and the parameter files for each instance. This chapter discusses how to devise a strategy for performing these tasks by explaining:

- Choosing a Backup Method
- Archiving the Redo Log Files
- Checkpoints and Log Switches
- Backing Up the Database

Oracle Parallel Server (OPS) supports all Oracle backup features in exclusive mode, including both open and closed backup of either an entire database or individual tablespaces.

Choosing a Backup Method

You can perform backup and recovery operations using two methods:

- Using Recovery Manager
- Using the operating system

The information provided in this chapter is true for both methods, unless specified otherwise.

Note: To avoid confusion between online and offline datafiles and tablespaces, this chapter uses the terms "open" and "closed" to indicate whether a database is available or unavailable during a backup. The term "whole backup" or "database backup" indicates that all datafiles and control files have been backed up. "Full" and "incremental" backups refer only to particular types of backups provided by Recovery Manager.

See Also: The *Oracle8i Backup and Recovery Guide* for a complete discussion of backup and recovery operations and terminology.

Archiving the Redo Log Files

This section explains how to archive the redo log files for each instance of a parallel server:

- Archiving Mode
- Automatic or Manual Archiving
- Archive File Format and Destination
- Redo Log History in the Control File
- Backing Up the Archive Logs

Archiving Mode

Oracle provides two archiving modes: ARCHIVELOG mode and NOARCHIVELOG mode. With Oracle in ARCHIVELOG mode, the instance must archive its redo logs as they are filled—before they can be overwritten. Oracle can then recover the log files in the event of media failure. In ARCHIVELOG mode, you can produce both open and closed backups. In NOARCHIVELOG mode, you can only make closed backups.

Note: Archiving is a per-instance operation that can be handled in one of two ways:

- Each instance on a parallel server can archive its own redo log files
- Alternatively, one or more instances can archive the redo log files manually for all instances, as described in the following section

See Also: "Open and Closed Database Backups" on page 21-12.

Automatic or Manual Archiving

Archiving can be performed automatically or manually for a given instance, depending on the value you set for the LOG_ARCHIVE_START initialization parameter.

- With LOG_ARCHIVE_START set to TRUE, Oracle automatically archives redo logs as they fill
- With LOG_ARCHIVE_START set to FALSE, Oracle waits until you instruct it to archive

You can set LOG_ARCHIVE_START differently for each OPS instance. For example, you can manually use SQL commands or Server Manager to have instance 1 archive the redo log files of instance 2, if instance 2 has LOG_ARCHIVE_START set to FALSE.

Automatic Archiving

The ARCH background process performs automatic archiving upon instance startup when LOG_ARCHIVE_START is set to TRUE. With automatic archiving, online redo log files are copied only for the instance performing the archiving.

In the case of a closed thread, the archiving process in the active instance performs the log switch and archiving for the closed thread. This is done when log switches are forced on all threads to maintain roughly the same range of SCNs in the archived logs of all enabled threads.

Manual Archiving

When LOG_ARCHIVE_START is set to FALSE, you can perform manual archiving in one of the following ways:

- Use the ARCHIVE LOG clause of the SQL command ALTER SYSTEM
- Enable automatic archiving using the SQL command ALTER SYSTEM ARCHIVE LOG START, or using Server Manager

Manual archiving is performed by the user process issuing the archiving command; it is not performed by the instance's ARCH process.

ALTER SYSTEM ARCHIVE LOG Options for Manual Archiving

ALTER SYSTEM ARCHIVE LOG manual archiving options include:

ALL	All online redo log files that are full but have not been archived.
CHANGE	The lowest system change number (SCN) in the online redo log file.
CURRENT	The current redo log of every enabled thread.
GROUP *integer*	The group number of an online redo log.
LOGFILE '*filename*'	The filename of an online redo log file in the thread.
NEXT	The next full redo log file that needs to be archived.
SEQ *integer*	The log sequence number of an online redo log file.
THREAD *integer*	The thread containing the redo log file to archive (defaults to the thread number assigned to the current instance).

You can use the THREAD option of ALTER SYSTEM ARCHIVE LOG to archive redo log files in a thread associated with an instance other than the current instance.

> **See Also:** "Forcing a Log Switch" on page 21-10 regarding threads and log switches. Refer to the *Oracle8i Reference* for information about the syntax of the ALTER SYSTEM ARCHIVE LOG statement. Also see the *Oracle8i Backup and Recovery Guide* as well as the "Archiving Redo Information" chapter in the *Oracle8i Administrator's Guide* for more information about manual and automatic archiving.

Monitoring the Archiving Process

The GV$ARCHIVE_PROCESSES and V$ARCHIVE_PROCESSES views provide information about the state of the various ARCH processes on the database and instance respectively. The GV$ARCHIVE_PROCESSES view displays 10*n rows, where 'n' is the number of open instances for the database. The V$ARCHIVE_PROCESSES view displays 10 rows, 1 row for each possible ARCH process.

> **See Also:** For more information about these views, please refer to the *Oracle8i Reference*.

Archive File Format and Destination

Archived redo logs are uniquely named as specified by the LOG_ARCHIVE_FORMAT parameter. This operating-system specific format can include text strings, one or more variables, and a filename extension. LOG_ARCHIVE_FORMAT can have variables as shown in Table 21–1. Examples in this table assume that LOG_ARCHIVE_FORMAT= arch%*parameter*, and the upper bound for all parameters is 10 characters.

Table 21–1 Archived Redo Log Filename Format Parameters

Parameter	Description	Example
%T	Thread number, left-zero-padded	arch0000000001
%t	Thread number, not padded	arch1
%S	Log sequence number, left-zero-padded	arch0000000251
%s	Log sequence number, not padded	arch251

The thread parameters %t and %T are used only with OPS. For example, if the instance associated with redo thread number 7 sets LOG_ARCHIVE_FORMAT to LOG_%s_T%t.ARC, then its archived redo log files are named:

```
LOG_1_T7.ARC
LOG_2_T7.ARC
LOG_3_T7.ARC
...
```

Note: Always specify thread and sequence number in archive log file format for easy identification of the redo log file.

See Also: The "Archiving Redo Information" chapter in the *Oracle8i Administrator's Guide* for information about specifying the archived redo log filename format and destination. Also refer to the "Recovery Structures" chapter in *Oracle8i Concepts.* Your Oracle system-specific documentation also contains information about the default log archive format and destination.

Redo Log History in the Control File

You can use the MAXLOGHISTORY clause of the CREATE DATABASE or CREATE CONTROLFILE command to enable the control file to keep a history of redo log files that a parallel server has filled. After creating the database, it is only possible to increase or decrease the log history by creating a new control file. Using CREATE CONTROLFILE destroys all log history in the current control file.

The MAXLOGHISTORY option specifies how many entries can be recorded in the archive history. Its default value is operating-system specific. If MAXLOGHISTORY is set to a value greater than zero, then whenever an instance switches from one online redo log file to another, its LGWR process writes the following data to the control file.

- Thread number
- Log sequence number
- Low system change number (SCN)
- Low SCN timestamp
- Next SCN (that is, the low SCN of the next log in sequence)

Note: LGWR writes log history data to the control file during a log switch, not when a redo log file is archived.

Log history records are small and are overwritten in a circular fashion when the log history exceeds the limit set by MAXLOGHISTORY.

During recovery, Server Manager prompts for the appropriate file names. Recovery Manager automatically restores the redo logs it requires. You can use the log history to reconstruct archived log file names from an SCN and thread number, for automatic media recovery of a parallel server that has multiple redo threads. An Oracle instance accessing the database in exclusive mode with only one thread enabled does not need the log history. However, the log history is useful when multiple threads are enabled even if only one thread is open.

You can query the log history information from the V$LOG_HISTORY view. When using Server Manager, V$RECOVERY_LOG also displays information about archived logs needed to complete media recovery. This information is derived from log history records.

Multiplexed redo log files do not require multiple entries in the log history. Each entry identifies a group of multiplexed redo log files, not a particular filename.

> **See Also:** "Restoring and Recovering Redo Log Files" on page 22-9 for Server Manager prompts that appear during recovery. Your Oracle system-specific documentation also has information about the default MAXLOGHISTORY value.

Backing Up the Archive Logs

Archive logs are generally only accessible by the node on which they were created. In OPS you have two backup options:

- Have each node back up its own archive logs
- Move the archive logs to one node, and then back them up

Use O/S utilities to manually implement either solution.

Node to Log Affinity

Optionally, you can set up each node to backup its own logs by running multiple copies of RMAN, one on each node. The new OEM (Oracle Enterprise Manager) architecture allows you to construct a single backup job and have it submitted to multiple nodes at times you specify.

Backing Up Archive Logs with Recovery Manager

Recovery Manager can automatically enable each node to back up its own archive logs. However, to move the logs you must do so manually and then use the

appropriate RMAN catalog and change commands to reflect the movement of files. Once Recovery Manager has been informed of the changes you have made, it can back up archive logs from the single node.

If you are using multiple nodes to back up your archive logs, when Recovery Manager compiles the list of logs to be archived, it must be able to check that the archived logs exist. To do this it must be able to read the headers of all archived logs on all nodes.

Each node can then back up the archived logs it has created. In the example below, because the initial target database is node 1 (on the RMAN command line), you must ensure that node 1 is able to read the headers of the archived logs (even those produced by node 2).

```
RMAN TARGET INTERNAL/KNL@NODE1 RCVCAT RMAN/RMAN@RCAT

RUN {
  ALLOCATE CHANNEL T1 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL T2 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  BACKUP
  FILESPERSET 10
  FORMAT 'AL_%T_%S_%P'
  (ARCHIVELOG UNTIL TIME 'SYSDATE' THREAD 1 DELETE INPUT CHANNEL T1)
  (ARCHIVELOG UNTIL TIME 'SYSDATE' THREAD 2 DELETE INPUT CHANNEL T2);
   )
```

Restoring Archive Logs with Recovery Manager

By default, RMAN restores archive logs to the *log_archive_dest* of the instances it connects to. If you are using multiple nodes to restore and recover, the archive logs may be restored to any of the nodes doing the restore / recover. The node actually reading the restored logs and performing the roll-forward is the target node initially connected to. To make recovery use these logs, ensure that the logs are readable from that node.

Using the CONNECT Option of the ALLOCATE CHANNEL Command

The CONNECT option of the RMAN ALLOCATE CHANNEL command allows you to allocate channels on any node of an OPS cluster. If you allocate channels on more than one node in the cluster, RMAN automatically distributes backup processing among those nodes.

On AIX and Pyramid Mesh clusters, RMAN also automatically detects disk-to-node affinity and backs up datafiles onto nodes that can most quickly access those datafiles.

Creating backups on multiple OPS nodes requires the following support from your media manager:

- The media manager must have a central catalog that any nodes can access.
- You must either have tape drives on every node you use, or the media manager must support backup / restore over the network.

Checkpoints and Log Switches

This section discusses:

- Checkpoints
- Forcing a Checkpoint
- Forcing a Log Switch
- Forcing a Log Switch on a Closed Thread

Checkpoints

Oracle8*i* performs checkpointing automatically on a consistent basis. Checkpointing requires that Oracle write all dirty buffers to disk and advance the checkpoint.

> **See Also:** For more information about checkpoints and how to control Oracle's checkpointing process, please refer to *Oracle8i Tuning*.

Forcing a Checkpoint

The SQL statement ALTER SYSTEM CHECKPOINT explicitly forces Oracle to perform a checkpoint for either the current instance or all instances. Forcing a checkpoint ensures that all changes to the database buffers are written to the datafiles on disk.

The GLOBAL option of ALTER SYSTEM CHECKPOINT is the default. It forces all instances that have opened the database to perform a checkpoint. The LOCAL option forces a checkpoint by the current instance.

A global checkpoint is not finished until all instances that require recovery have been recovered. If any instance fails during the global checkpoint, however, the checkpoint might complete before that instance has been recovered.

To force a checkpoint on an instance running on a remote node, you can change the current instance with the Server Manager command CONNECT.

Note: You need the ALTER SYSTEM privilege to force a checkpoint.

See Also: "Specifying Instances" on page 18-17 for information on specifying a remote node.

Forcing a Log Switch

A parallel server can force a log switch for any instance that fails to archive its online redo log files for some period of time, either because the instance has not generated many redo entries or because the instance has shut down. This prevents an instance's redo log, known as a *thread* of redo, from remaining unarchived for too long. If media recovery is necessary, the redo entries used for recovery are always reasonably recent.

For example, after an instance has shut down, another instance can force a log switch for that instance so its current redo log file can be archived.

The SQL statement ALTER SYSTEM SWITCH LOGFILE forces the current instance to begin writing to a new redo log file, regardless of whether the current redo log file is full.

Forcing all instances to perform log switches is known as a *global log switch.* To do this, use the SQL statement ALTER SYSTEM ARCHIVE LOG CURRENT and omit the THREAD keyword. After issuing this statement, Oracle waits until all online redo log files are archived before returning control to you. Use this statement to force a single instance to perform a log switch and archive its online redo log files by specifying the THREAD keyword.

In Server Manager, use the Instance Force Log Switch option for the current instance only. There is no global option for forcing a log switch in Server Manager. You may want to force a log switch so that you can archive, drop, or rename the current redo log file.

Note: You need the ALTER SYSTEM privilege to force a log switch.

See Also: "Redo Log Files" on page 6-3 for more information about threads.

Forcing a Log Switch on a Closed Thread

You can force a closed thread to complete a log switch while the database is open. This is useful if you want to drop the current log of the thread. This procedure does not work on an open thread, including the current thread, even if the instance that had the thread open is shut down. For example, if an instance aborted while the thread was open, you could not force the thread's log to switch.

To force a log switch on a closed thread, manually archive the thread, using the Begin Manual Archive dialog box of Server Manager or the SQL command ALTER SYSTEM with the ARCHIVE LOG option. For example:

```
ALTER SYSTEM ARCHIVE LOG GROUP 2;
```

To archive a closed redo log group manually that will force it to log switch, you must connect with SYSOPER or SYSDBA privileges.

> **See Also:** The *Oracle8i Administrator's Guide* for information on connecting with SYSDBA or SYSOPER privileges.

Backing Up the Database

This section covers backup operation issues in OPS. It covers the following topics:

- Open and Closed Database Backups
- Recovery Manager Backup Issues
- Operating System Backup Issues

Open and Closed Database Backups

All backup operations can be performed from any node of a parallel server. Open backups allow you to back up all or part of the database while it is running. Users can access the database and update data in any part of the database during an open backup. With a parallel server you can make open backups of multiple tablespaces simultaneously from different nodes. An open backup includes copies of one or more datafiles and the current control file. Subsequent archived redo log files or incremental backups are also necessary to allow recovery up to the time of a media failure.

When using the operating system, closed backups are taken while the database is closed. When using Recovery Manager, an instance must be started and mounted, but not open, to do a closed backup. Before making a closed backup, shut down *all* instances of a parallel server. While the database is closed, you can back up its files in parallel from different nodes. A closed whole database backup includes copies of all datafiles and the current control file.

If you archive redo log files, a closed backup allows recovery up to the time of a media failure. In NOARCHIVELOG mode, full recovery is not possible since a closed backup only allows restoration of the database to the point in time of the backup.

Warning: Do not use operating-system utilities to back up the control file in ARCHIVELOG mode unless you are performing a whole, closed backup.

Never erase, reuse, or destroy archived redo log files until completing another whole backup, or preferably two whole backups, either open or closed.

> **See Also:** The *Oracle8i Backup and Recovery Guide* and the chapters "Database Backup" and "Database Recovery" in *Oracle8i Concepts*.

Recovery Manager Backup Issues

Preparing for Snapshot Control Files in Recovery Manager

In OPS, you must prepare for snapshot control files before you perform a backup using Recovery Manager.

Any node making a backup may need to create a snapshot control file. Therefore, on all nodes used for backup, you must ensure the existence of the directory to which such a snapshot control file will be written.

For example, to specify that the snapshot control file should be written to the file **/oracle/db_files/snapshot/snap_prod.cf**, you would enter:

```
SET SNAPSHOT CONTROLFILE TO '/ORACLE/DB_FILES/SNAPSHOT/SNAP_PROD.CF';
```

You must then ensure that the directory **/oracle/db_files/snapshot** exists on all nodes from which you perform backups.

It is also possible to specify a raw device destination for a snapshot control file, which like other datafiles in OPS will be shared across all nodes in the cluster.

Performing an Open Backup Using Recovery Manager

See the *Oracle8i Backup and Recovery Guide* for complete information on open backups using Recovery Manager.

If you are also backing up archive logs, then issue an ALTER SYSTEM ARCHIVE LOG CURRENT statement after the backup has completed. This ensures that you have all redo to make the files in this backup consistent.

The following sample script distributes datafile and archive log backups across two instances in a parallel server environment. It assumes:

- There are more than 20 files in the database
- 4 tape drives available, two on each node
- The archive log files produced by thread 2 are readable by node1

The sample script is as follows:

```
RUN {
  ALLOCATE CHANNEL NODE1_T1 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL NODE1_T2 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL NODE2_T3 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  ALLOCATE CHANNEL NODE2_T4 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  BACKUP
```

```
  FILESPERSET 6
  FORMAT 'DF_%T_%S_%P'
  (DATABASE);
SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
BACKUP
  FILESPERSET 10
  FORMAT 'AL_%T_%S_%P'
  (ARCHIVELOG UNTIL TIME 'SYSDATE' LIKE 'node1_archivelog_dest%' DELETE
   INPUT CHANNEL NODE1_T1)
  (ARCHIVELOG UNTIL TIME 'SYSDATE' LIKE 'node2_archivelog_dest%' DELETE
   INPUT CHANNEL NODE2_T3);
```

Node Affinity Awareness

On some cluster platforms, certain nodes of the cluster have faster access to some datafiles than to other datafiles. RMAN automatically detects such affinity. When deciding which channel will back up a particular datafile, RMAN gives preference to channels allocated at nodes with affinity to that datafile. To use this feature, allocate RMAN channels at the various nodes of the cluster that have affinity to the datafiles being backed up.

For example:

```
RUN
{
  ALLOCATE CHANNEL CH1 TYPE 'SBT_TAPE' CONNECT '@INST1';
  ALLOCATE CHANNEL CH2 TYPE 'SBT_TAPE' CONNECT '@INST2';
  ...
}
```

Note: For more information about the CONNECT operand of the ALLOCATE command, please see the *Oracle8i Backup and Recovery Guide*.

Operating System Backup Issues

This section discusses the following backup issues:

- Beginning and Ending an Open Backup Using Operating System Utilities
- Performing an Open Backup Using Operating System Utilities

Beginning and Ending an Open Backup Using Operating System Utilities

When using the operating system method, you can begin an open backup of a tablespace at one instance and end the backup at the same instance or another instance. For example:

```
ALTER TABLESPACE TABLESPACE BEGIN BACKUP;/* INSTANCE X */
Statement processed.

....OPERATING SYSTEM COMMANDS TO COPY DATAFILES...
....COPY COMPLETED...

ALTER TABLESPACE TABLESPACE END BACKUP;/* INSTANCE Y */
Statement processed.
```

> **Note:** If the ALTER TABLESPACE ... BEGIN BACKUP command is not issued or does not complete before an operating system backup of the tablespace is started, then the backed up datafiles are not useful for subsequent recovery operations. Attempting to recover such a backup is risky and can cause errors resulting in inconsistent data.

It does not matter which instance issues each of these statements, but they must be issued whenever you make an open backup. The BEGIN BACKUP option has no effect on users' access to the tablespace.

For an open backup to be usable for complete or incomplete media recovery, you must retain all archived redo logs spanning the period of time between the execution of the BEGIN BACKUP command and the recovery end-point.

After making an open backup, you can force a global log switch by using ALTER SYSTEM ARCHIVE LOG CURRENT. This statement archives all online redo log files that need to be archived, including the current online redo log files of all

enabled threads and closed threads of any instance that shut down without archiving its current redo log file.

> **See Also:** The *Oracle8i SQL Reference* for a description of the BEGIN BACKUP and END BACKUP clauses of the ALTER TABLESPACE command.

Performing an Open Backup Using Operating System Utilities

The following steps are recommended if you are using operating system utilities to perform an open backup in OPS.

1. Before starting the open backup, issue the ALTER SYSTEM ARCHIVE LOG CURRENT command.

   This switches and archives the current redo log file for *all* threads in OPS, even threads that are not currently up.

2. Issue the ALTER TABLESPACE *tablespace* BEGIN BACKUP command.
3. Wait for the ALTER TABLESPACE command to successfully complete.
4. In the operating-system environment, issue the appropriate commands to back up the datafiles for the tablespace.
5. Wait for the operating-system backup to successfully complete.
6. Issue the ALTER TABLESPACE *tablespace* END BACKUP command.
7. Back up the control files with ALTER DATABASE BACKUP CONTROLFILE TO *filename*.

For an added measure of safety, back up the control file to a trace file with the ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS command, then identify and back up that trace file.

If you are also backing up archive logs, then issue an ALTER SYSTEM ARCHIVE LOG CURRENT statement after END BACKUP. This ensures that you have all redo to roll to the end backup marker.

22

Recovering the Database

This chapter describes Oracle recovery features on a parallel server. It covers the following topics:

- Overview
- Recovery from Instance Failure
- Recovery from Media Failure
- Parallel Recovery

Overview

This chapter discusses three types of recovery:

Table 22–1 Types of Recovery

Type of Recovery	Definition
Instance failure.	Occurs when a software or hardware problem prevents an instance from continuing work.
Media failure.	Occurs when the storage medium for Oracle files is damaged. This usually prevents Oracle from reading or writing data.
Parallel recovery.	For Recovery Manager, the restore and application of incremental backups are parallelized using channel allocation. Application of redo (whether it is done by Recovery Manager or by Server Manager) is determined by the RECOVERY_PARALLELISM parameter.

Recovery from Instance Failure

The following sections describe the recovery performed after failure of instances accessing the database in shared mode.

- Single-node Failure
- Multiple-node Failure
- Fast-Start Checkpointing
- Fast-Start Roll Back
- Access to Datafiles for Instance Recovery
- Freezing the Database for Instance Recovery
- Phases of Oracle Instance Recovery

After instance failure, Oracle uses the online redo log files to perform automatic recovery of the database. For a single instance running in exclusive mode, instance recovery occurs as soon as the instance starts up again after it has failed or shut down abnormally.

When instances accessing the database in shared mode fail, online instance recovery is performed automatically. Instances that continue running on other nodes are not affected as long as they are reading from the buffer cache. If instances attempt to write, the transaction stops. All operations to the database are suspended until cache recovery of the failed instance is complete.

See Also: The *Oracle8i Backup and Recovery Guide.*

Single-node Failure

Oracle Parallel Server (OPS) performs instance recovery by coordinating recovery operations through the SMON processes of the other running instances. If one instance fails, the SMON process of another instance notices the failure and automatically performs instance recovery for the failed instance.

Instance recovery does not include restarting the failed instance or any applications that were running on that instance. Applications that were running may continue by failover, as described in "Recovery from Instance Failure" on page 22-2.

When one instance performs recovery for another failed instance, the surviving instance reads redo log entries generated by the failed instance and uses that information to ensure all committed transactions are reflected in the database. Data from committed transactions is not lost. The instance performing recovery rolls

back any transactions that were active at the time of the failure and releases resources being used by those transactions.

Multiple-node Failure

As long as one instance continues running, its SMON process performs instance recovery for any other instances that fail in a parallel server.

If all instances of a parallel server fail, instance recovery is performed automatically the next time an instance opens the database. The instance does not have to be one of the instances that failed, and it can mount the database in either shared or exclusive mode from any node of the parallel server. This recovery procedure is the same for Oracle running in shared mode as it is for Oracle in exclusive mode, except that one instance performs instance recovery for all failed instances.

Fast-Start Checkpointing

Fast-start checkpointing is the basis for Fast-start fault recovery in Oracle8*i*. Fast-start checkpointing occurs continuously, advancing the checkpoint as Oracle write blocks to disk. Fast-start checkpointing always writes the oldest modified block first, ensuring that every write allows the checkpoint time to be advanced. This eliminates bulk writes and the resulting I/O spikes that occur with conventional checkpointing, yielding smooth and efficient on-going performance.

You can specify a limit on how long the roll forward phase of Fast-start checkpointing takes. Oracle automatically adjusts the checkpoint write rate to meet the specified roll-forward limit while issuing the minimum number of writes. For details on how to do this, please refer to *Oracle8i Tuning*.

Fast-Start Roll Back

The rollback phase of system fault recovery in Oracle8*i* uses "non-blocking" rollback technology. This means new transactions can begin immediately after roll forward completes. When a new transaction accesses a row locked by a dead transaction, the new transaction rolls back only the changes that prevent the transaction's progress. New transactions do not have to wait for Oracle to roll back the entire dead transaction, so long-running transactions no longer affect recovery time. The Fast-start technology maximizes data availability and ensures predictable recovery time.

In addition, the database server can roll back dead transactions in parallel. This technique is used against rows not blocking new transactions, and only when the

cost of performing dead transaction roll back in parallel is less than performing it serially.

See Also: *Oracle8i Concepts.*

Access to Datafiles for Instance Recovery

An instance performing recovery for another instance must have access to all online datafiles that the failed instance was accessing. When instance recovery fails because a datafile fails verification, the instance that attempted to perform recovery does not fail but a message is written to the ALERT file.

After you correct the problem that prevented access to the database files, use the SQL statement ALTER SYSTEM CHECK DATAFILES to make the files available to the instance.

See Also: "Datafiles" on page 6-2.

Freezing the Database for Instance Recovery

With OPS, you can use the dynamic parameter FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY to control freezing of the database during instance recovery. All instances must have the same value for this parameter.

When this parameter is set to TRUE, Oracle freezes the entire database during instance recovery. The advantage of freezing the entire database is to stop other disk activities except those for instance recovery. Instance recovery may thus complete sooner. The drawback of freezing the entire database is that it becomes unavailable during instance recovery.

When this parameter is set to FALSE, Oracle does not freeze the database, thus part of the unaffected database is accessible during instance recovery.

The system attempts to intelligently select an appropriate default.

- If all online datafiles use hash locks, the default value of this parameter is FALSE. This is because when hash locks are used most parts of the database can be accessed by users during instance recovery.
- If data files use fine grain locks, the default is TRUE. When fine grain locks are used an instance death may affect a larger portion of the database. Affected data will be accessible only after instance recovery. In this case, setting this parameter to TRUE can potentially make those parts of the database available sooner.

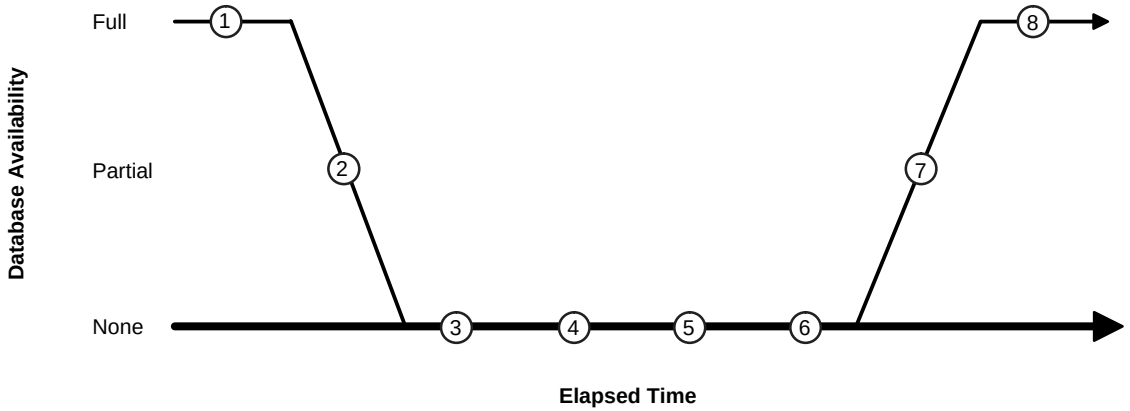
To see the number of times the entire database is frozen for instance recovery after this instance has started up, check the "instance recovery database freeze count" statistic in V$SYSSTAT.

See Also: The *Oracle8i Reference*.

Phases of Oracle Instance Recovery

Figure 22–1 illustrates the degree of database availability during each phase of Oracle instance recovery.

Figure 22–1 Phases of Oracle Instance Recovery



Phases of recovery are these:

1. OPS is running on multiple nodes.
2. Node failure is detected.
3. The LM is reconfigured; resource and lock management is redistributed onto the set of surviving nodes. One call gets persistent resources. Lock value block is marked as dubious for locks held in exclusive or protected write mode. Lock requests are queued.
4. LCK*n* processes build a list of all invalid lock elements.
5. Roll forward. Redo logs of the dead thread(s) are applied to the database.
6. LCK*n* processes make all invalid lock elements valid.

7. Roll back. Rollback segments are applied to the database for all uncommitted transactions.
8. Instance recovery is complete, and all data is accessible.

During phase 5, forward application of the redo log, database access is limited by the transitional state of the buffer cache. The following data access restrictions exist for all user data in all datafiles, regardless of whether you are using hashed or fine grain locking, or any particular features:

- No writes to surviving buffer caches can succeed while the access is limited.
- No disk I/O of any sort by way of the buffer cache and direct path can be done from any of the surviving instances.
- No lock requests are made to the IDLM for user data.

Reads of buffers already in the cache with the correct global lock can be done, since they do not involve any I/O or lock operations.

The transitional state of the buffer cache begins at the conclusion of the initial lock scan phase when instance recovery is first started by scanning for dead redo threads. Subsequent lock scans are made if new "dead" threads are discovered. This state lasts while the redo log is applied (cache recovery) and ends when the redo logs have been applied and the file headers have been updated. Cache recovery operations conclude with validation of the invalid locks, which occurs after the buffer cache state is normalized.

Recovery from Media Failure

After a media failure resulting in the loss of one or more database files, use backups of the datafiles to recover the database.

If you are using Recovery Manager, you might also need to apply incremental backups, archived redo log files and a backup of the control file.

If you are using operating system utilities, you might need to apply archived redo log files to the database and use a backup of the control file.

This section describes:

- Complete Media Recovery
- Incomplete Media Recovery
- Restoring and Recovering Redo Log Files
- Disaster Recovery

See Also: The *Oracle8i Backup and Recovery Guide* for procedures to recover from various types of media failure.

Complete Media Recovery

You can perform complete media recovery in either exclusive or shared mode. Table 22–2 shows the status of the database that is required to recover particular database objects.

Table 22–2 Database Status for Media Recovery

To Recover	Database Status
An entire database or the SYSTEM tablespace.	The database must be mounted but not opened by any instance.
A tablespace other than the SYSTEM tablespace.	The database must be opened by the instance performing the recovery and the tablespace must be offline.
A datafile.	The database can be open with the datafile offline, or the database can be mounted but not opened by any instance. (For a datafile in the SYSTEM tablespace, the database must be mounted but not open.)

You can recover multiple datafiles or tablespaces on multiple instances simultaneously.

Complete Media Recovery Using Operating System Utilities

With operating system utilities you can perform open database recovery of tablespaces or datafiles in shared mode. Do this using the Server Manager command RECOVER TABLESPACE or RECOVER DATAFILE.

You can use the Server Manager RECOVER DATABASE command to recover a database that is mounted in shared mode, but not open. Only one instance can issue this command in OPS.

Note: The recommended method of recovering a database is to use Server Manager. We do not recommend direct use of the SQL command ALTER DATABASE RECOVER.

Complete Media Recovery Using Recovery Manager

With Recovery Manager you can issue the following statements to restore and recover the files:

- RESTORE DATABASE
- RESTORE TABLESPACE
- RESTORE DATAFILE
- RECOVER DATABASE
- RECOVER TABLESPACE
- RECOVER DATAFILE

The commands you use in Recovery Manager for OPS are the same as those you use to recover single instance environments.

> **See Also:** For more information refer to the *Oracle8i Backup and Recovery Guide*.

Incomplete Media Recovery

Incomplete media recovery can be performed while the database is mounted in shared or exclusive mode but not opened by any instance. Do this using the following database recovery options:

With Recovery Manager use one of the following options with the SET command prior to restoring and recovering:

- UNTIL CHANGE *integer*
- UNTIL TIME *date*
- UNTIL LOGSEQ *integer* THREAD *integer*

With operating system utilities restore your appropriate backups and then use one of the following options with the RECOVER DATABASE command:

- UNTIL CANCEL
- UNTIL CHANGE *integer*
- UNTIL TIME *date*

> **See Also:** The *Oracle8i Backup and Recovery Guide.*

Restoring and Recovering Redo Log Files

Media recovery of a database accessed by OPS may require multiple archived log files to be open at the same time. Because each instance writes redo log data to a separate redo thread, recovery may require as many as one archived log file per thread.

However, if a thread's online redo log contains enough recovery information, restoring archived log files for that thread is unnecessary.

Recovery Using Recovery Manager

Recovery Manager automatically restores and applies the archive logs required. By default, Recovery Manager restores archive logs to the LOG_ARCHIVE_DEST directory of the instances to which it connects. If you are using multiple nodes to restore and recover, this means that the archive logs may be restored to any of the nodes performing the restore / recover.

The node that actually reads the restored logs and performs the roll forward is the target node to which the connection was initially made. You must ensure that the logs are readable from that node.

> **See Also:** The *Oracle8i Backup and Recovery Guide* for information about overriding the location to which Recovery Manager restores archive logs.

Recovery Using Operating System Utilities

During recovery, Oracle prompts you for the archived log files as they are needed. Messages supply information about the required files and Oracle prompts you for the filenames.

For example, if the log history is enabled and the filename format is LOG_T%t_SEQ%s, where %t is the thread and %s is the log sequence number, then you might receive these messages to begin recovery with SCN 9523 in thread 8:

```
ORA-00279: Change 9523 generated at 27/09/91 11:42:54 needed for thread 8
ORA-00289: Suggestion : LOG_T8_SEQ438
ORA-00280: Change 9523 for thread 8 is in sequence 438
Specify log: {<RET> = suggested | filename | AUTO | FROM | CANCEL}
```

If you use the ALTER DATABASE statement with the RECOVER clause instead of Server Manager, you receive these messages but not the prompt. Redo log files may be required for each enabled thread in OPS. Oracle issues a message when a log file

is no longer needed. The next log file for that thread is then requested, unless the thread was disabled or recovery is finished.

If recovery reaches a time when an additional thread was enabled, Oracle simply requests the archived log file for that thread. Whenever an instance enables a thread, it writes a redo entry that records the change; therefore, all necessary information about threads is available from the redo log files during recovery.

If recovery reaches a time when a thread was disabled, Oracle informs you that the log file for that thread is no longer needed and does not request further log files for the thread.

Note: If Oracle reconstructs the names of archived redo log files, the format that LOG_ARCHIVE_FORMAT specifies for the instance doing recovery must be the same as the format specified for the instances that archived the files. All instances should use the same value of LOG_ARCHIVE_FORMAT in OPS, and the instance performing recovery should also use that value. You can specify a different value of LOG_ARCHIVE_DEST during recovery if the archived redo log files are not at their original archive destinations.

Disaster Recovery

This section describes disaster recovery using Recovery Manager and operating system utilities. *Disaster recovery* is used when a failure makes an entire site unavailable. In this case, you can recover at an alternate site using open or closed database backups.

Note: To recover up to the latest point in time, all logs must be available at a remote site; otherwise some committed transactions may be lost.

Disaster Recovery Using Recovery Manager

The following scenario assumes:

- You have lost the entire database, all control files and the online redo log
- You will be distributing your restore over 2 nodes
- There are 4 tape drives (two on each node)

- You are using a recovery catalog

> **Note:** It is highly advisable to back up the database immediately after opening the database reset logs, since all previous backups are invalidated. This step is not shown in this example.

The SET UNTIL command is used in case the database structure has changed in the most recent backups and you wish to recover to that point in time. In this way, Recovery Manager restores the database to the same structure the database had at the specified time.

Before You Begin: Before beginning the database restore, you must:

- Restore your initialization file and your recovery catalog from your most recent backup
- Catalog archive logs, datafile copies, or backup sets that are on disk but are not registered in the recovery catalog

  The archive logs up to the logseq number being restored *must* be cataloged in the recovery catalog, or Recovery Manager will not know where to find them.

  If you resynchronize the recovery catalog frequently, and have an up-to-date copy from which you have restored, there should not be many archive logs that need cataloging.

> **Note:** You only have to perform this step if you lose your recovery catalog and have already restored and performed point-in-time recovery on it. This is not necessary if the recovery catalog is still intact. You might, however, need to catalog a few archived logs, even with an intact catalog, but you only need to recreate the ones that were created since the last "catalog resync". A "catalog resync" is the process by which rman copies information about backups, copies, and archivelogs from the target database control file to the recovery catalog.

What the Sample Script Does: The following script restores and recovers the database to the most recently available archived log, which is log 124 thread 1. It does the following:

- Starts the database NOMOUNT and restricts connections to DBA-only users.

- Restores the control file to the location specified.
- Copies (or replicates) this control file to all the other locations specified by the CONTROL_FILES initialization parameter.
- Mounts the control file.
- Catalogs any archive logs not in the recovery catalog.
- Restores the database files (to the original locations).

  If volume names have changed, you must use the statement SET NEWNAME FOR... before the restore, then perform a switch after the restore. This updates the control file with the datafiles' new locations.
- Recovers the datafiles by either using a combination of incremental backups and redo, or just redo.

  Recovery Manager completes the recovery when it reaches the log sequence number specified.
- Opens the database resetlogs.

> **Note:** Only complete the following step if you are certain there are no other archived logs to apply.

- Oracle recommends you back up your database after the resetlogs. This is not shown in the example.

Restore/Recover Sample Script:

The DBA starts Server Manager as follows:

```
CONNECT SCOTT/TIGER AS SYSDBA
```

Oracle responds with:

```
Connected.
```

Then enter the following STARTUP syntax:

```
STARTUP NOMOUNT RESTRICT
```

The DBA starts Recovery Manager and runs the script.

Note: The user specified in the target parameter must have SYSDBA privilege.

```
RMAN TARGET SCOTT/TIGER@NODE1 RCVCAT RMAN/RMAN@RCAT
RUN {
  SET UNTIL LOGSEQ 124 THREAD 1;
  ALLOCATE CHANNEL T1 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL T2 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE1';
  ALLOCATE CHANNEL T3 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  ALLOCATE CHANNEL T4 TYPE 'SBT_TAPE' CONNECT 'INTERNAL/KNL@NODE2';
  ALLOCATE CHANNEL D1 TYPE DISK;
  RESTORE CONTROLFILE;
  ALTER DATABASE MOUNT;
  CATALOG ARCHIVELOG '/ORACLE/DB_FILES/NODE1/ARCH/ARCH_1_123.RDO';
  CATALOG ARCHIVELOG '/ORACLE/DB_FILES/NODE1/ARCH/ARCH_1_124.RDO';
  RESTORE DATABASE;
  RECOVER DATABASE;
  SQL 'ALTER DATABASE OPEN RESETLOGS';
  }
```

Disaster Recovery Using Operating System Utilities

To do this, use the following procedure:

1. Restore the last full backup at the alternate site as described in the *Oracle8i Backup and Recovery Guide.*
2. Start Server Manager.
3. Connect as SYSDBA.
4. Start and mount the database with the STARTUP MOUNT statement.
5. Initiate an incomplete recovery using the RECOVER command with the appropriate UNTIL option.

   The following command is an example:

   ```
   RECOVER DATABASE USING BACKUP CONTROLFILE UNTIL CANCEL
   ```

6. When prompted with a suggested redo log file name for a specific thread, use that filename.

If the suggested archive log is not in the archive directory, specify where the file can be found. If redo information is needed for a thread and a file name is not suggested, try using archive log files for the thread in question.

7. Repeat step 6 until all archive log files have been applied.
8. Stop the recovery operation using the CANCEL command.
9. Issue the ALTER DATABASE OPEN RESETLOGS statement.

Note: If any distributed database actions are used, check to see whether your recovery procedures require coordinated distributed database recovery. Otherwise, you may cause logical corruption to the distributed data.

Parallel Recovery

The goal of the parallel recovery feature is to use compute and I/O parallelism to reduce the elapsed time required to perform crash recovery, single-instance recovery, or media recovery. Parallel recovery is most effective at reducing recovery time when several datafiles on several disks are being recovered concurrently.

Parallel Recovery Using Recovery Manager

With Recovery Manager's RESTORE and RECOVER commands Oracle can automatically parallelize all three stages of recovery.

Restoring Data Files: When restoring data files, the number of channels you allocate in the Recovery Manager recover script effectively sets the parallelism Recovery Manager uses. For example, if you allocate 5 channels, you can have up to 5 parallel streams restoring data files.

Applying Incremental Backups: Similarly, when you are applying incremental backups, the number of channels you allocate determines the potential parallelism.

Applying Redo Logs: Oracle applies the redo logs in parallel as determined by the RECOVERY_PARALLELISM parameter.

The RECOVERY_PARALLELISM initialization parameter specifies the number of redo application server processes participating in instance or media recovery. One process reads the log files sequentially and dispatches redo information to several recovery processes that apply the changes from the log files to the datafiles. A value of 0 or 1 indicates recovery is to be performed serially by one process. The value of

this parameter cannot exceed the value of the PARALLEL_MAX_SERVERS parameter.

Parallel Recovery Using Operating System Utilities

You can parallelize instance and media recovery two ways:

- Setting the RECOVERY_ PARALLELISM Parameter
- Specifying RECOVER Command Options

The Oracle Server can use one process to read the log files sequentially and dispatch redo information to several recovery processes to apply the changes from the log files to the datafiles. Oracle automatically starts the recovery processes, so you do not need to use more than one session to perform recovery.

Setting the RECOVERY_ PARALLELISM Parameter

The RECOVERY_PARALLELISM initialization parameter specifies the number of redo application server processes participating in instance or media recovery. One process reads the log files sequentially and dispatches redo information to several recovery processes. The recovery processes then apply the changes from the log files to the datafiles. A value of 0 or 1 indicates that recovery is performed serially by one process. The value of this parameter cannot exceed the value of the PARALLEL_MAX_SERVERS parameter.

Specifying RECOVER Command Options

When you use the RECOVER command to parallelize instance and media recovery, the allocation of recovery processes to instances is operating system specific. The DEGREE keyword of the PARALLEL clause can either signify the number of processes on each instance of a parallel server or the number of processes to spread across all instances.

> **See Also:** *Oracle8i Concepts* for more information on Fast-start Parallel Rollback and your Oracle system-specific documentation for more information on the allocation of recovery processes to instances.

Fast-start Parallel Rollback in OPS

Setting the INIT.ORA parameter FAST_START_PARALLEL_ROLLBACK to LOW or HIGH enables Fast-start Parallel Rollback. This parameter helps determine the maximum number of server processes that participate in Fast-start parallel rollback. If the value is FALSE, Fast-start parallel rollback is disabled.

If the value for FAST_START_PARALLEL_ROLLBACK is LOW, the number of processes used for Fast-start rollback is 2 times the value for CPU_COUNT. If the value is HIGH, at most 4 times the value of CPU_COUNT is the number of rollback servers used for Fast-start parallel rollback.

In OPS, multiple parallel recovery processes are owned by and operated only within the instance that generated them. To determine an accurate setting for FAST_START_PARALLEL_ROLLBACK, examine the contents of V$FAST_START_SERVERS and V$FAST_START_TRANSACTIONS.

Fast-start Parallel Rollback does not perform cross-instance rollback. However, it can improve the processing of rollback segments for a single database with multiple instances since each instance can spawn its own group of recovery processes.

Managed Standby and Standby Databases

You can protect OPS systems against disasters by using standby databases. To simplify the adminstration of standby databases, consider using the Managed Standby feature. Please refer to the *Oracle8i Backup and Recovery Guide* for details about the Managed Standby Database feature.

23

Migrating from a Single Instance to Parallel Server

This chapter describes database conversion: how to convert from a single instance Oracle database to a multi-instance Oracle database using the Oracle Parallel Server (OPS) option.

The chapter is organized as follows:

- Overview
- Deciding to Convert
- Preparing to Convert
- Converting the Database from Single- to Multi-instance
- Troubleshooting the Conversion

Overview

This chapter explains how to enable your database structure to support multiple instances. It also explains how to begin a project with a single instance Oracle database even though you intend to migrate to the multi-instance parallel server. In addition, this chapter can help you extend an existing OPS configuration to additional nodes.

Note: Before using this chapter to convert to a multi-instance database, use the *Oracle8i Migration* manual to perform any necessary upgrade of the Oracle Server. That manual also provides information on upgrading and downgrading in replicated systems.

Deciding to Convert

This section describes:

- Reasons to Convert
- Reasons Not to Convert

Reasons to Convert

You may decide to convert to a multi-instance database for the following reason:

- To move from a single node to a cluster (when you have designed your application with OPS in mind).

In addition, your application may have been designed for OPS but you need more instances to take advantage of your current database design. Or you may have enough nodes but need to bring offline nodes online. You might even already be using OPS but want to add more nodes.

Reasons Not to Convert

Do not convert to OPS in the following situations:

- You are using a file system that is not shared.
- Your application was not designed for parallel processing; you need to examine your application more.
- You are not using a supported configuration (of shared disks, and so on).

Preparing to Convert

This section describes:

- Hardware and Software Requirements
- Converting the Application from Single- to Multi-instance
- Administrative Issues

Hardware and Software Requirements

To convert to OPS you must have:

- A supported hardware and OS software configuration
- A license for the Oracle Parallel Server
- Oracle Server running on all nodes
- OPS linked to your system

Converting the Application from Single- to Multi-instance

Making your database run in parallel does not automatically mean you have effectively implemented OPS. Besides migrating your existing database from single instance Oracle to multi-instance Oracle, you must also migrate existing applications that were designed for single-instance Oracle. Preparing an application for use with a multi-instance database may require application partitioning and physical schema changes.

> **See Also:** Chapter 12, "Application Analysis" for a full discussion of this topic.

Administrative Issues

Note the following administrative issues of conversion:

- Your backup procedures should be in place before converting from single-instance Oracle to the Oracle Parallel Server.
- Additional archiving considerations apply in an OPS environment. In particular, the archive file format needs the thread number. Furthermore, archived logs from all nodes are needed for media recovery. If you archive to a file, then on systems where file systems cannot be shared, some method of accessing the archive logs is required.

> **See Also:** Chapter 21, "Backing Up the Database".

Converting the Database from Single- to Multi-instance

The following procedure explains how to migrate an existing database from single-instance Oracle to multi-instance Oracle. Remember that you must also migrate the application from single- to multi-instance.

1. Modify your application to make it OPS-ready.
2. Make sure all necessary files are shared among the nodes.

   OPS assumes disks are shared among instances such that each instance can access all log files, control files, and database files. These files should normally be on raw devices, since the disks are shared through raw devices on most clusters.

   Note: You cannot use NFS to share files for OPS. NFS does not provide adequate availability: if the node goes down, NFS goes down and the files cannot be reached. Moreover, NFS does not provide adequate consistency: a write may be cached and not written to disk immediately.

3. Check MAXINSTANCES on the single instance.

   The MAXINSTANCES parameter was set at database creation, usually to its default value of 1. With MAXINSTANCES set to 1, only one instance can run on database, and the database cannot run in parallel server mode. The number of rows in V$THREAD is one per created thread. The MAXINSTANCES value may be much higher. You can check V$ACTIVE_INSTANCES to find this value.

   To check the value of MAXINSTANCES query V$ACTIVE_INSTANCES. Alternatively, you can dump the control file to a trace file by entering:

   ```
   ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
   ```

   The trace file may look like this:

   ```
   Dump file /mf1/qjones/qj1/rdbms/log/ora_20016.trc
   Oracle8 Server Release 8.0.3
   With the distributed, replication, parallel query and
      Parallel Server options
   PL/SQL Release 3.0
   ORACLE_HOME = /mf1/qjones/qj1
   ORACLE_SID = mf1qj1
   Oracle process number: 19        Unix process id: 20016
   System name:    mf1seq
   ```

```
Node name:      mf1seq
Release:        3.2.0
Version:        V2.1.1
Machine:        i386
Wed Feb 22 14:30:22 1997
Wed Feb 22 14:30:23 1997
*** SESSION ID:(18.1)
# The following commands will create a new control file and
# use it to open the database.
# No data other than log history will be lost. Additional logs
# may be required for media recovery of offline data files.
# Use this only if the current version of all online logs are
# available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "TPCC" NORESETLOGS
NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 62
     MAXINSTANCES 1
    MAXLOGHISTORY 100
LOGFILE
  GROUP 1 '/dev/rvol/v-qj80W-log11'  SIZE 200M,
  GROUP 2 '/dev/rvol/v-qj80W-log12'  SIZE 200M
DATAFILE
  '/dev/rvol/v-qj80W-sys',
  '/dev/rvol/v-qj80W-temp',
  '/dev/rvol/v-qj80W-cust1',
.
.
.
;
# Recovery is required if any of the datafiles are restored
# backups, or if the last shutdown was not normal or
# immediate.
RECOVER DATABASE
# Database can now be opened normally.
ALTER DATABASE OPEN;
```

4. Edit the control file script to include a larger MAXINSTANCES value.

Edit the trace file so it only contains SQL commands to generate the CREATE CONTROLFILE statement. Then make the following changes:

- Set PFILE to point to the correct initialization file.

- Increase the MAXINSTANCES parameter to the number of Oracle instances you want to support.
- Use a large value for the MAXLOGHISTORY parameter.

The resulting control file is a script that recovers and reopens your database if necessary.

Before running the SQL file, make sure the current control file(s) are in the backup directory.

A sample script follows:

```
STARTUP NOMOUNT PFILE=$HOME/perf/tkvc/admin/tkvcrun.ora
CREATE CONTROLFILE REUSE DATABASE "TPCC" NORESETLOGS
NOARCHIVELOG
    MAXLOGFILES 16
    MAXLOGMEMBERS 2
    MAXDATAFILES 62
    MAXINSTANCES 1
    MAXLOGHISTORY 100
LOGFILE
  GROUP 1 '/dev/rvol/v-qj80W-log11'  SIZE 200M,
  GROUP 2 '/dev/rvol/v-qj80W-log12'  SIZE 200M
DATAFILE
  '/dev/rvol/v-qj80W-sys',
  '/dev/rvol/v-qj80W-temp',
  '/dev/rvol/v-qj80W-cust1',
.
.
.
;
# Recovery is required if any of the datafiles are restored
# backups, or if the last shutdown was not normal or
# immediate.
RECOVER DATABASE
# Database can now be opened normally.
ALTER DATABASE OPEN;
```

5. Back up the new control file immediately after conversion. We also recommend beginning your backup procedures for the database.
6. Decide how to administer the initialization parameter file(s).

   Each instance has private initialization parameters. However, some of the parameters need to have the same value on each instance. There are two ways of administering this.

One approach is for each instance to have a private parameter file that includes the common parameter file shared between the instances. The common parameter file must be on a shared device accessible by all nodes. This way, when you need to make a generic change to one of the common initialization parameters, you need only make the change on one node rather than on all nodes.

Alternatively, you can make multiple copies of the parameter file and place one on the private disk of each node of your OPS environment. In this case, you must update all parameter files each time you make a generic change.

7. Edit the following parameters in the instance-specific initialization parameter file:

   a. Specify an INSTANCE_NUMBER for this instance. Each instance will be numbered at startup time. The instance number is used in the free list group assignment. If you do not specify the INSTANCE_NUMBER, Oracle assigns a number based on start up order.

   b. Specify ROLLBACK_SEGMENTS. Each instance should have a set of private rollback segments.

   c. Specify the THREAD parameter in the initialization parameter file so the instance always starts with the same set of redo log files. A thread number is assigned at startup time to associate an instance with the log files of that thread. By default this value is 0; you can set it to 1 for the first instance.

   d. Add the DB_NAME parameter to the initialization parameter file.

8. Make sure the following common initialization parameters have identical values for all instances:

   CONTROL_FILES
   DB_BLOCK_SIZE
   DB_FILES
   DB_NAME
   GC_FILES_TO_LOCKS
   GC_ROLLBACK_LOCKS
   LM_LOCKS (identical values recommended)
   LM_PROCS (identical values recommended)
   LM_RESS (identical values recommended)
   MAX_COMMIT_PROPAGATION_DELAY
   ROW_LOCKING
   SINGLE_PROCESS

9. Make sure the Oracle executable is linked with the OPS option and that each node is running the same versions of the executable. The banner you see upon connection should display the words "Parallel Server".

   Note: Corruption may occur if one node opens the database in shared mode and another node opens it in exclusive mode.

10. Perform a normal shutdown of the database.
11. Back up the control files using operating system commands.
12. Remove the control files but retain backups of them.
13. Run the new script you built that recreates the old control files with new data—larger structures for some of the database objects.
14. Add rollback segments.
15. Add additional threads.
16. Shut down the database.
17. Start up the database in shared mode. The first instance will be started.
18. Add the second instance in shared mode, using the standard procedure described in "Starting in Shared Mode" on page 18-15. (Note that the second instance only succeeds if the first instance is in shared mode.) Add redo log files, rollback segments, and so on.
19. Tune the GC_* and LM_* parameters for optimal performance.

Troubleshooting the Conversion

This section explains how to resolve common errors:

- Database Recovery After Conversion
- Loss of Rollback Segment Tablespace
- Inadvisable NFS Mounting of Parameter File

Database Recovery After Conversion

If you should lose your database and Oracle8 files after converting from single-instance Oracle to OPS, restore your cold backup and then apply all changes from the redo logs. In this case, your old control file would be used as though you had never done the conversion. You would have to recreate the new control file if you migrate to OPS.

Loss of Rollback Segment Tablespace

The following problem may occur if a user has created tablespaces for private rollback segments and allocated them to specific instances at startup. It may also occur if files containing rollback segments are lost.

If you lose one rollback segment tablespace or file containing rollback segments due to media failure, all instances will fail. To recover, shut down all instances. All other rollback segments must remain offline so you can bring the one you want to recover off line.

Inadvisable NFS Mounting of Parameter File

As mentioned earlier, it is not advisable to access a common parameter file (or any Oracle file or executable) over NFS. If the NFS disk were to go down, no other instance could start. Access to control files and data files is not supported over NFS.

Part V

Reference

A

Differences Among Versions

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

- Differences Between 8.0.4 and 8.1
- Differences Between Release 8.0.3 and Release 8.0.4
- Differences Between Release 7.3 and Release 8.0.3
- Differences Between Release 7.2 and Release 7.3
- Differences Between Release 7.1 and Release 7.2
- Differences Between Release 7.0 and Release 7.1
- Differences Between Version 6 and Release 7.0

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

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 OPS 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

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 Rolback." 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 OPS. The installer collects node information from the user, distributes the required Oracle products to the specified nodes, and invokes the OPS Assistant to set up the instances and create the database.

When OPS 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 OPS environment.

Use this release 8.1 feature to improve load balancing and limit block pinging. For instance, using OPS 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 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 specifically 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 views 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

See Also: "Determining the Amount of Locks Needed and Setting LM_* Parameters" on page 18-12.

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

See Also: "GC_* Global Cache Parameters" on page 18-9.

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

See Also: "GC_* Initialization Parameters" on page 9-13.

Dynamic Performance Views

The following new views were added:

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.

See Also: "Global Dynamic Performance Views" on page 20-12.

Integrated 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.

See Also: Chapter 8, "Integrated Distributed Lock Manager".

Instance Groups

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

See Also: "Specifying Instance Groups" on page 18-23.

Group Membership Services

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

See Also: "The Cluster Manager" on page 18-22.

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.

See Also: "Recovery from Instance Failure" on page 22-2.

Recovery Manager

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

See Also: "Recovery from Media Failure" on page 22-6.

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.

Fine grain locking is discussed in the section "Two Methods of PCM Locking: Fixed and Releasable" on page 9-15.

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 query 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 OPS 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 Tuning* 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 OPS.

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 OPS 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 OPS 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 Tuning*.

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 Query Processor Affinity

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

Parallel query 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 query 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 datafile
- Taking a datafile 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
Dropping a tablespace	DROP TABLESPACE tablespace
Taking a tablespace offline or online	ALTER TABLESPACE tablespace OFFLINE ALTER TABLESPACE tablespace ONLINE
Adding a datafile	ALTER TABLESPACE tablespace ADD DATAFILE
Renaming a datafile	ALTER TABLESPACE tablespace RENAME DATAFILE
Renaming a datafile 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 datafile 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 datafile 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 datafile, create a tablespace, or drop a tablespace and its datafiles, 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 datafiles (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.

- Compatibility
- Restrictions

Compatibility

The following sections describe aspects of compatibility between shared and exclusive modes on a parallel server:

- The Export and Import Utilities
- Compatibility Between Shared and Exclusive Modes

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

OPS 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 OPS.

If OPS 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
- Restrictions in Shared Mode

Maximum Number of Blocks Allocated at a Time

The *!blocks* option of the GC_FILES_TO_LOCKS parameter enables you to control the number of blocks available for use within a free list group. You can use *!blocks* to specify the rate 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, as described in "Starting Instances" on page 18-13.

Maximum Number of Datafiles

The number of datafiles supported by Oracle is operating system specific. Within this limit, the maximum number allowed depends on the values used in the CREATE DATABASE command, 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 OPS 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

OPS 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 datablocks.

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

A

absolute file number, 6-3
acquiring rollback segments, 14-5
  initialization parameters, 6-9
acquisition AST, 8-2, 8-4
ADD LOGFILE clause
  THREAD clause, 6-3
  thread required, 14-8
ADDED_EXTENTS, A-10
adding a file, 14-9, 15-10, A-18
affinity
  awareness, 21-14
  disk, 4-8, 18-23
  parallel processor, A-15
ALERT file, 6-2, 22-4
ALL option, 21-4
ALL_TABLES table, 16-7
ALLOCATE EXTENT option
  DATAFILE option, 11-11, 17-12
  in exclusive mode, 17-11
  instance number, 11-15, 17-13
  INSTANCE option, 11-11, 17-12
  not available, A-20
  pre-allocating extents, 17-14
  SIZE option, 11-11, 17-11
allocation
  automatic, 11-17, 17-13, 17-14
  extents, 17-14, 17-15, 18-16
  free space, 11-6, 11-11
  of channels, 21-8
  PCM locks, 9-6, 9-26, 11-15, 17-11
  rollback segments, 14-5
  sequence numbers, 6-6
ALLOW_PARTIAL_SN_RESULTS parameter
  obsolete for 8.1, A-5
ALTER CLUSTER statement, A-20
  ALLOCATE EXTENT option, 17-11
  allocating extents, 17-14
ALTER DATABASE OPEN RESETLOGS statement, 22-14
ALTER DATABASE statement
  ADD LOGFILE, 6-3
  adding or dropping log file, A-18
  CLOSE option, 18-18
  DATAFILE OFFLINE and ONLINE options, A-18
  DATFILE RESIZE, 15-9
  DISABLE, 14-9
  ENABLE THREAD, 14-8
  MOUNT option, 18-13
  OPEN option, 18-14
  RECOVER, 22-7
  RECOVER option, 22-7
  renaming a file, A-18
  setting the log mode, 14-2, 14-9, B-2
  THREAD, 14-9
  thread of redo, 14-9
ALTER INDEX statement
  DEALLOCATE UNUSED option, 17-16
ALTER ROLLBACK SEGMENT command, 6-10
ALTER SESSION statement
  SET INSTANCE option, 11-11, 17-10
ALTER SYSTEM
  command, limiting instances for parallel execution, 18-27
ALTER SYSTEM ARCHIVE LOG statement, 18-18, 21-16

CURRENT option, 21-10, A-19
global log switch, 21-10, 21-16
THREAD option, 18-18, 21-4
ALTER SYSTEM CHECK DATAFILES statement, 6-2
instance recovery, 22-4
ALTER SYSTEM CHECKPOINT statement, 21-9
global versus local, 18-18
specifying an instance, 18-18
ALTER SYSTEM privilege, 21-10
ALTER SYSTEM SWITCH LOGFILE statement, 18-18, 21-10, A-19
DBA privilege, 21-10, A-19
ALTER TABLE statement
ALLOCATE EXTENT option, 17-11
allocating extents, 17-14, 17-15
DISABLE TABLE LOCK option, 7-6, 10-3, 16-7
ENABLE TABLE LOCK option, 7-6, 10-3, 16-7
MAXEXTENTS option, 17-15
ALTER TABLESPACE statement
ADD DATAFILE, 15-10
ADD DATAFILE option, A-18
BACKUP option, 21-15
OFFLINE and ONLINE options, A-18
READ ONLY option, 12-2
renaming a data file, A-18
applications
analysis of, 12-1
availability, 22-2
business functions, 12-2
compute-intensive, 1-21
converting to multi-instance, 23-3
departmentalized, 2-8
designing, 13-1
disjoint data, 1-17, 2-7
DSS, 1-5, 1-12, 1-21, 2-7
failover of, 4-11
insert-intensive, 11-12
node, 5-4
OLTP, 1-5, 1-12, 2-7
performance, 11-12
portability, 1-5
profile, 12-2
profiles, 3-2
query-intensive, 1-17, 2-7
random access, 2-8
redesigning for parallel processing, 1-16
scalability, 2-2, 2-6
tuning, 12-1
tuning performance, 1-15, 11-12
ARCH process, 5-5, 21-4
architecture
hardware, 3-1
multi-instance, 5-4
Oracle database, 6-1
Oracle instance, 5-1
archive log
backup, 21-7
ARCHIVE LOG clause
CURRENT option, 21-10, 21-16, A-19
global log switch, 21-10, 21-16
manual archiving, 21-4
specifying an instance, 18-21
THREAD option, 21-4
ARCHIVE LOG command, 21-3
ARCHIVELOG mode, 14-9
automatic archiving, 4-6, 21-3
changing mode, 14-2, 14-9, B-2
creating a database, 14-2
online and offline backups, 4-6, 21-3
archiving redo log files, 21-1
automatic versus manual, 21-3
conversion to multi-instance, 23-3
creating a database, 14-2
forcing a log switch, 21-10
history, 21-6
identified in control file, 6-5
log sequence number, 21-5
online archiving, 4-6, 21-3
AST, 8-2
asymmetrical multiprocessing, 2-5
asynchronous trap, 8-2, 8-3, 8-4
authentication
password file, 18-26
AUTOEXTEND, 15-9
automatic archiving, 21-3
automatic recovery, 21-7
availability, 12-1
and interconnect, 3-3
benefit of parallel databases, 1-14

data files, 6-2, 22-4
multiple databases, 1-19
phases of recovery, 22-5
shared disk systems, 3-6
single-node failure, 2-12, 22-2

B

background process, 5-5
ARCH, 21-4
holding instance lock, 7-4, 7-6
instance, 5-4
LGWR, 21-6
optional, 5-5
parallel cache management, 4-12
SMON, 18-27, 22-2
backup, 21-1
archive log, 21-7
conversion to multi-instance, 23-3
export, B-1
files used in recovery, 22-7
offline, 4-6, 21-12
online, 4-6, 21-12, 21-16
parallel, 21-12
bandwidth, 1-11, 2-2, 2-12, 3-3, 3-4, 3-5, 3-6
batch applications, 1-12, 1-13
BEGIN BACKUP option, 21-15
block
allocating dynamically, 17-15
associated with instance, 11-11
cleanout, A-14
contention, 6-9, 11-15, 15-6, 17-11, 17-13
deferred rollback, 6-9, A-19
distributed lock, 5-5
free space, 11-2
instance lock, 4-12
multiple copies, 4-5, 4-12, 5-5
segment header, 11-14
when written to disk, 4-5, 9-8
Block Server Process
definition, 1-17, 5-7
blocking AST, 8-3
blocks
associated with instance, 22-3
BSP
definition, 1-17, 5-7
buffer cache, 5-5
coherency, 4-12
distributed locks, 2-7
instance recovery, 22-3
minimizing I/O, 4-5, 4-12
written to disk, 4-5
buffer state, 9-10
buffer, redo log, 5-5

C

cache
buffer cache, 4-5, 5-5
coherency, 4-12, 9-2
consistency, on MPPs, 3-8
dictionary cache, 4-12, 5-5, 6-6
flushing dictionary, 4-16
management issues, non-PCM, 4-16
parallel cache management, 4-12
recovery, 22-6
row cache, 6-6
sequence cache, 4-7, 6-7
Cache Fusion
and cache coherency conflicts, 20-2
architecture, 5-7
benefits, 20-4
change summary for 8.1, A-1
definition, 4-6, 5-7
description, 20-2
performance, 20-1
tuning, 20-1
CACHE keyword, 18-28
CACHE option, CREATE SEQUENCE, 6-6
CACHE_SIZE_THRESHOLD parameter, A-17
obsolete for 8.1, A-5
capacity planning, 19-3
CATPARR.SQL script, 15-12, 20-12
channels
allocating for backup and recovery, 21-8
CHECK DATAFILES clause, 6-2
instance recovery, 22-4
checkpoint
forcing, 21-9
CHECKPOINT_PROCESS parameter, 18-10

CKPT process, 5-5
CLEANUP_ROLLBACK_ENTRIES
parameter, A-9, A-13
client connections to services, 18-28
client load balancing, 18-32
client-server configuration, 5-4
description, 1-20
Net8, 1-20
closed thread, 21-4, 21-16
cluster
allocating extents, 17-14
free list groups, 17-11
free lists, 11-12, 17-7
hash cluster, 11-12, 17-7, A-20
implementations, 2-3
performance, 3-2
version compatibility, 18-22
Cluster Manager (CM) software, 18-13, 18-14
purpose, 18-22
committed data
checkpoint, 21-9
instance failure, 22-2
sequence numbers, 6-7
compatibility
shared and exclusive modes, 6-2, 17-11
concurrency
inserts and updates, 11-14, 17-6
maximum number of instances, 11-15, 14-3
sequences, 6-7
shared mode, 4-2
transactions, 5-4, 11-2
configurations
change in redo log, 14-9
client-server, 1-20
guidelines for parallel server, 5-9
multi-instance database system, 1-16
overview of Oracle, 1-14, 1-15
single instance database system, 1-15
CONNECT
RMAN ALLOCATE CHANNEL
command, 21-8
CONNECT command, 14-7, 18-19, 18-21
forcing a checkpoint, 21-9
Net8, 18-17
SYSDBA option, 18-13, 18-26
connect string, 18-17, 18-20, 18-21
connect time failover, 18-32
connections
clients and services, 18-28
load balancing among nodes, 18-32
consistency
multiversion read, 4-6
rollback information, 6-8
contention
block, 6-9, 11-15, 15-6, 17-11, 17-13
disk, 6-2, 6-9
distributed lock, 15-6
free list, 19-5
free space, 4-8, 11-2, 11-14
index, 19-6
on single block or row, 2-9
rollback segment, 6-8, 6-9
segment header, 11-14, 19-6
sequence number, 4-7, 6-6
SYSTEM tablespace, 14-5
table data, 6-2, 6-8, 17-13
control file, 6-1
accessibility, 5-9
backing up, 21-1
conversion to multi-instance, 23-6
creating, 14-10
data files, 17-12
log history, 14-4, 21-6
MAXLOGHISTORY, 6-5
parameter values, 18-9
shared, 5-4, 18-7
CONTROL_FILES parameter, 18-11, 22-12
same for all instances, 18-7, 18-11
conversion
application, 23-3
database, 23-4
database, to multi-instance, 23-1
ramifications, 23-3
convert queue, 8-3
CPU usage, 19-3
CPU_COUNT parameter, 18-11
CR Server
change summary for 8.1, A-1
CREATE CLUSTER statement, 17-7, A-20
FREELIST GROUPS option, 17-6

FREELISTS option, 17-6
CREATE CONTROLFILE statement
changing database options, 14-10
conversion to multi-instance, 23-5
exclusive mode, B-2
MAXLOGHISTORY, 21-6
CREATE DATABASE statement, 14-3
exclusive mode, B-2
MAXINSTANCES, 11-15, 14-3
MAXLOGFILES, 14-4
MAXLOGHISTORY, 6-5, 14-4, 21-6
MAXLOGMEMBERS, 14-4
options, 14-3
setting the log mode, 14-2, 14-9
thread number 1, 14-8
CREATE INDEX statement
FREELISTS option, 17-8
CREATE ROLLBACK SEGMENT statement, 14-5, 14-6
CREATE SEQUENCE statement, 6-7
CACHE option, 6-6, 6-7
CYCLE option, 6-6
description, 6-6
ORDER option, 6-6, 6-7
CREATE TABLE statement
clustered tables, 17-7
examples, 17-14
FREELIST GROUPS option, 11-11
FREELISTS option, 11-11, 17-6
initial storage, 17-13, 17-14
CREATE TABLESPACE statement, A-18
creating a rollback segment, 14-5, 14-6
creating a tablespace, A-18
current instance, 18-21
checkpoint, 21-9
log switch, 21-10
specifying, 18-17
CURRENT option, 21-4
checkpoints, 21-10
forcing a global log switch, 21-10
global log switch, 21-16
new in Oracle7, A-19
CYCLE option, CREATE SEQUENCE, 6-6

D

data block, 5-5
data dependent routing, 12-7, 19-6
data dictionary, 5-5
objects, 6-6
querying views, 20-12
row cache, 6-6
sequence cache, 6-7
views, 14-6
data warehousing, 2-7
database
archiving mode, 14-2, 14-9
backup, 4-6, 21-1
closing, 18-26
conversion to multi-instance, 23-1, 23-4
creation, 14-3
designing, 1-15, 13-1
dismounting, 18-26
export and import, B-1
migrating to multi-instance, 23-3
mounted but not open, 14-9
number of archived log files, 6-5, 21-6
number of instances, 11-15, 14-3
rollback segments, 14-5
scalability, 2-5
standby, 18-33, 22-16
starting NOMOUNT, 22-11
database administrator (DBA)
distributed database, 1-19
data-dependent routing, 3-8
datafile
accessibility, 5-9
adding, 15-6, 15-10, 15-12, A-18
allocating extents, 17-12
backing up, 21-1
disk contention, 6-2
dropping, A-18
file ID, 15-3
instance recovery, 6-2, 22-4
mapping locks to blocks, 9-6
maximum number, B-3
multiple files per table, 11-15, 17-11, 17-13
number of blocks, 15-3
parallel recovery, 22-7

recovery, 22-7, A-22
renaming, A-18
shared, 5-4, 6-2
tablespace, A-18
tablespace name, 15-3
taking offline or online, A-18
unspecified for PCM locks, 9-6
validity, 15-12
DATAFILE option
table, 17-14
tablespace, A-18
DB_BLOCK_BUFFERS parameter, 9-18
ensuring LM lock capacity, 16-6
GC_RELEASABLE_LOCKS, 15-13
DB_BLOCK_SIZE parameter
same for all instances, 18-11
DB_FILES parameter
calculating non-PCM resources, 16-4
ensuring LM lock capacity, 16-6
same for all instances, 18-11
DB_NAME parameter
conversion to multi-instance, 23-7
same for all instances, 18-11
DBA_ROLLBACK_SEGS view, 6-8, 14-7
public rollback segments, 14-6
DBA_SEGMENTS view, 14-7
DBA_TABLES table, 16-7
DBMS_JOB package
and instance affinity, 4-9
DBMS_SPACE package, 17-16
DBMSUTIL.SQL script, 17-16
DBWR process, 5-5, 7-6
in parallel server, 5-2
DDL
commands, 16-6
lock, 7-5
deadlock detection, 8-9
deallocating unused space, 17-16
ded, vi
default instance, 18-17, 18-20
deferred rollback segment, A-19
DELAYED_LOGGING_BLOCK_CLEANOUTS parameter, 18-10, A-9, A-14
departmentalized applications, 2-8
DESCRIPTION_LIST feature, A-13
dictionary cache, 4-12, 5-5, 6-6
lock, 10-5
dictionary cache lock, 10-5
DISABLE TABLE LOCK option, 16-7
DISABLE THREAD clause, 14-9
disabling the archive history, 14-4
disaster recovery, 22-10, 22-13
DISCONNECT command, 18-21
disconnecting from an instance, 18-20, 18-21
multiple sessions, 18-27
user process, 18-26
disjoint data
applications with, 1-17, 2-7
data files, 6-2
disk
access, 3-2, 3-3
affinity, 4-8, 18-23
contention, 6-2, 6-9
I/O statistics, 19-3, 19-6
media failure, 22-1
reading blocks, 4-5
rollback segments, 6-9
writing blocks, 4-5, 9-8
distributed lock
memory area, 5-5
rollback segment, 6-9
row cache, 6-6
sequence, 6-6
total number, 6-6
Distributed Lock Manager, A-8
DM, Database Mount, 10-5
DML lock, 7-3, 7-5
DML_LOCKS parameter, 7-6, 10-3, 18-10, 18-11
and performance, 16-7
calculating non-PCM resources, 16-4
ensuring IDLM lock capacity, 16-6
DROP TABLE command, 4-16
DROP TABLESPACE statement, A-18
dropping a database object
tablespace, 15-6, A-18
dropping a redo log file, A-18
log switch, 21-10
manual archiving, 14-9
restrictions, 14-9
DSS applications, 1-5, 1-12, 1-13, 1-21, 2-7, 3-2

dual ported controllers, 2-3
dynamic performance view
  creating, 20-12
dynamically allocating blocks, 11-17

E

ENABLE TABLE LOCK option, 16-7
END BACKUP option, 21-15
enqueue, 7-3
  global, 7-3
  in V$LOCK, 7-9
  local, 7-3
  OPS context, 7-6
enqueue locks
  calculating non-PCM resources, 16-4
ENQUEUE_DEBUG_MULTI_INSTANCE
    parameter (Oracle Version 6), A-21
ENQUEUE_RESOURCES parameter
  calculating non-PCM resources, 16-4
error message
  parameter values, 18-16
  rollback segment, 14-5
  storage options, 17-6
exclusive mode, 8-7, 18-12
  compatibility, B-1
  database access, 1-15, 4-1
  free lists, 17-6, 17-11
  MAXLOGHISTORY, 21-7
  media recovery, 14-4
  required for file operations, A-17, B-2
  specifying instance number, 17-13
  specifying thread number, 18-15
  startup, 17-13, 18-12
  switching archive log mode, 14-9
  taking tablespace offline, A-18
EXCLUSIVE option, 18-13
EXCLUSIVE parameter
  obsolete for 8.0.4, A-6
exclusive PCM lock, 4-15
Export utility
  and free lists, 11-12, A-20, B-3
  backing up data, B-1
  compatibility, B-1
EXT_TO_OBJ table, 15-12, 20-12
extent
  allocating PCM locks, 11-15, 17-11
  allocating to instance, 17-10, 17-14, 18-16
  definition, 11-3
  initial allocation, 17-13
  not allocated to instance, 11-6, 11-17, 17-12
  rollback segment, 6-8, 14-7
  size, 6-8, 14-7, 17-11
  specifying a file, 17-12

F

failover, 4-11, 16-2, A-11
  connect time, 18-32
failure
  access to files, 22-4
  ALERT file, 6-2
  instance recovery, 22-4
  media, 22-1, 22-7, A-22
  MPP node, 3-7
  node, 1-19, 22-2
false pings, 9-17, 15-15
FAST_START_PARALLEL_ROLLBACK, A-3
Fast-start Parallel Rollback
  parallel rollback, Fast-start, 22-16
Fast-start Rollback, A-3
fault tolerance, 8-8
file
  adding, A-18
  ALERT, 6-2, 22-4
  allocating extents, 17-12
  archiving redo log, 4-6, 21-3, 21-4, 21-5
  common parameter file, 18-6
  control file, 6-1, 6-5, 21-6
  datafile, 6-1
  dropping, 14-9, 21-10, A-18
  exported, B-1
  maximum number, B-3
  multiplexed, 21-7
  number, absolute, 6-3
  number, relative, 6-3
  parameter, 18-3, 18-7
  PFILE, 18-6, 18-8
  redo log, 4-6, 6-3, 21-3, 21-5, 21-6, A-18
  renaming, 14-9, 21-10, A-18

restricted operations, 21-10, A-17
size, 9-6
used in recovery, 22-7
FILE_LOCK view, 9-22, 15-11, A-9
fine grain lock, 9-4, 9-6, 9-16, 9-18, 9-19, 9-20, 9-21
creation, 9-3
DBA lock, 9-17
group factor, 15-9
introduction, 7-5
one lock element to one block, 9-17
specifying, 15-9
fine grain locking, 2-8, A-9
fixed hashed PCM lock, 9-3
fixed mode, lock element, 9-20
flexibility of parallel database, 1-14
FORCE parameter
and job-to-instance affinity, 4-10
foreground process
instance shutdown, 18-26
format, lock name, 7-8
free list, 11-15
and Export utility, 11-12, B-3
assigned to instance, 11-13
cluster, 17-7, A-20
concurrent inserts, 4-8, 11-14
contention, 19-5
definition, 11-5
exclusive mode, B-2
extent, 11-12
hash cluster, 17-7
in exclusive mode, 17-6, 17-11
index, 17-8
list groups, 11-12
number of lists, 17-6
partitioning, 11-13
partitioning data, 11-12, 18-16
PCM locks, 11-15, 17-11
transaction, 11-4
unused space, 17-16
free list group
assigned to instance, 11-13, 11-14
assigning to session, 17-10
definition, 11-5
enhanced for release 7.3, A-9
high performance feature, 4-8
setting !blocks, 15-8
unused space, 17-16
used space, 17-16
FREE_LIST_INST parameter (Oracle Version 6), A-20, A-21
FREE_LIST_PROC parameter (Oracle Version 6), A-20, A-21
FREED_EXTENTS, A-10
FREELIST GROUPS storage option, 17-6, 17-14, 18-14
clustered tables, A-20
instance number, 11-15
FREELISTS
parameter, 11-5
storage option, 17-6
FREELISTS storage option
clustered tables, A-20
indexes, 17-8
maximum value, 17-6
FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY parameter, 22-4, A-6

G

GC_DB_LOCKS parameter, A-6
adjusting after file operations, A-18
GC_FILES_TO_LOCKS parameter, 9-3, 9-4, 9-6, 9-14, 9-24, 14-10, 15-17, 18-11
adding datafiles, 15-12
adjusting after file operations, 15-6, A-18
associating PCM locks with extents, 11-15, 17-11
default bucket, 15-7
fine grain examples, 15-9
guidelines, 15-9
hashed examples, 15-8
index data, 15-5
reducing false pings, 15-16
room for growth, 15-10
setting, 15-6
syntax, 15-7
GC_FREELIST_GROUPS parameter, A-6, A-9
GC_LATCHES parameter, A-5
obsolete for 8.1, A-5
GC_LCK_PROCS parameter
obsolete for 8.1, A-5

GC_RELEASABLE_LOCKS parameter, 9-14, 15-17, A-9
default, 15-13
GC_ROLLBACK_LOCKS parameter, 6-9, 9-14, 9-15, 15-7, 15-13, 15-17, 18-11
GC_ROLLBACK_SEGMENTS parameter, A-6
number of distributed locks, 6-9
GC_SAVE_ROLLBACK_LOCKS parameter, 6-9, A-6, A-19
GC_SEGMENTS parameter, A-6
GC_SORT_LOCKS parameter, A-21
GC_TABLESPACES parameter, A-7
global constant parameter, 18-11
and non-PCM locks, 4-12
control file, 6-1
description, 9-13
list of, 18-9
rollback segments, 6-9
same for all instances, 18-9, 18-11
global dynamic performance view, 18-25, 20-12, A-7
GLOBAL hint, 20-12
GLOBAL option
forcing a checkpoint, 18-18, 21-9
verifying access to files, 6-2
GMS
removed for 8.1, A-2
granted queue, 8-3, 8-4
group
free list, 11-12
MAXLOGFILES, 14-4
redo log files, 6-4, 14-4, 14-9
unique numbers, 6-5
V$LOGFILE, 6-5
Group Membership Services, A-8
removed for 8.1, A-2
GROUP option, 21-4
group-based locking, 8-9, 8-10
growth, room for, 15-10
GV$ view, 18-25, 20-12, A-7
GV$BH view, 19-5
GV$CACHE view, 19-5, 20-12
GV$CLASS_PING view, 20-12
GV$FILE_PING view, 20-12
GV$PARAMETER view, 18-25
GV$PING view, 19-5, 20-12

H

hardware
architecture, 3-1
requirements, 3-3
scalability, 2-2
hash cluster, 17-7
free lists, 11-12, A-20
hashed PCM lock, 9-4, 9-17, 9-21, 9-22
creation, 9-3
introduction, 7-5
releasable, 9-4, 15-7, 15-9
specifying, 15-8
hashing
static, lock mastering scheme, 8-9
header
rollback segment, 14-7
segment, 11-14, 14-7
high availability
benefit of parallel databases, 1-14
high speed interconnect, 12-1
high water mark, 11-17
definition, 11-3, 11-18
moving, 11-18, 11-19
high-speed bus, 3-5, 3-6
history, archive, 21-6, 22-9
horizontal partitioning, 2-12
HOST command, 18-19
host, IDLM, 9-8

I

identifier, lock, 7-8
IDLM, 8-1
IDLM parameters, 18-12
IFILE parameter, 18-5
multiple files, 18-7
overriding values, 18-7
specifying identical parameters, 18-6
Import utility
Compatibility, B-1
free lists, A-20
restoring data, B-1

incremental growth, 17-13
index
contention, 19-6
creating, 17-8
data partitioning, 11-12, 15-5
FREELISTS option, 17-8
PCM locks, 15-5
INITIAL storage parameter
minimum value, 17-13
rollback segments, 6-8
initialization parameter
archiving, 21-3
control of blocks, 9-13
control of PCM locks, 9-13
displaying values, 18-16, 18-21
duplicate values, 18-7
global constant, 6-1, 18-9
guidelines, 5-9
identical for all instances, 18-11
Integrated Distributed Lock Manager, 18-12
MAX_DEFAULT_PROPAGATION_DELAY, 18-11
obsolete, A-21
PARALLEL_DEFAULT_MAXSCANS, 18-11
planning LM capacity, 16-3, 16-6
using default value, 18-7
inserts
concurrent, 4-8, 11-14, 17-6
free lists, 11-14, 18-16
free space unavailable, 17-11
performance, 11-12
instance
adding instances, 14-4, 17-13, 23-2
affinity, for jobs, 4-9, A-4
associated with data block, 11-11
associated with data file, 17-13
associated with extent, 17-10
background processes, 4-12, 5-4, 5-5
changing current, 18-21
changing default, 18-20
current, 18-20, 18-21, 21-9
failure, 22-3
free list, 11-14, 17-11
groups, specifying, 18-23
groups, using, 18-24
instance number, 11-15, 17-13
maximum number, 6-8, 11-15, 14-3, 14-4
number, 17-10
ownership of PCM locks, 9-5
parallel server characteristics, 5-1
recovery, 14-4, 18-27, 22-2
remote, 18-6, 18-8, 18-20
rollback segment required, 6-8
startup order, 18-16
thread number, 6-3, 14-8, 18-15
instance lock, 7-2
acquired by background process, 7-6
acquired by foreground process, 7-6
definition, 7-2, 7-4
types, 7-5
INSTANCE NAME parameter, 18-6
instance number, 11-13
INSTANCE option
allocating, 17-14
SET INSTANCE command, 17-10, 18-17
SHOW INSTANCE command, 18-20
instance recovery
abnormal shutdown, 18-27
access to files, 6-2, 22-4
global checkpoint, 21-9
multiple failures, 22-3
rollback segments, 6-8
starting another instance, 14-4
instance registration, A-10
purpose, 18-28
INSTANCE_GROUPS parameter, 18-24
INSTANCE_ID column, 20-12
INSTANCE_NUMBER parameter, 17-10, 18-5
and SQL options, 11-11
assigning free lists to instances, 11-13
conversion to multi-instance, 23-7
exclusive mode, 18-14
exclusive or shared mode, 18-16
individual parameter files, 18-5
setting, 17-13
unique values for instances, 18-10, 18-16
unspecified, 18-16
INSTANCES keyword, 18-27
INSTANCES parameter (Oracle Version 6), A-21
Integrated Distributed Lock Manager

capacity for locks and resources, 16-2
configuring, 16-3
definition, 1-11
distributed architecture, 8-8
failover requirements, 16-2
fault tolerant, 8-8
features, 8-8
group-based locking, 8-10
handling lock requests, 8-3
instance architecture, 5-2
internalized, A-8
LCKn process, 9-8
lock mastering, 8-9
minimizing use, 1-16, 4-12
non-PCM lock capacity, 7-6
queues, 8-2
recovery phases, 22-5
resource sharing, 9-8
integrated operations, 1-5
interconnect, 3-3
and scalability, 2-2
high-speed, 1-9
protocols for OPS, 20-6
INTERNAL option
instance shutdown, 18-26
inter-node communication, 1-9
I/O
and scalability, 2-3
disk contention, 4-5
interrupts, 2-5
minimizing, 1-16, 4-5, 4-12, 15-6
IPCs
and Cache Fusion, 20-6

J

jobs
and instance affinity, 4-9

L

Lamport SCN generation, 4-7
LANGUAGE parameter (Oracle Version 6), A-21
latch, 7-3, 10-5
latency, 1-11, 1-13, 2-2, 2-12, 3-3
LCKn process, 5-5, 9-5
on multi-instance database, 1-16
role in recovery, 22-5
LGWR process, 5-5, 7-6
log history, 21-6
library cache lock, 10-4
links, 5-9
listener
in instance registration, 18-31
load balancing, A-3
LM_LOCKS parameter, 15-17, 15-18, 16-2, 18-11, 18-12, A-6
LM_PROCS parameter, 18-11, 18-12, A-6
LM_RESS parameter, 15-17, 15-18, 16-2, 16-5, 18-11, 18-12, A-6
LMDn process, 5-5, A-8
definition, 7-7
LMON process, 5-5, A-8
definition, 7-7
load balancing, 12-5, A-13
client connections among listeners, 18-32
in parallel execution, 18-33
of connections among nodes, 18-32
LOAD_BALANCE parameter
configuring client load balancing, 18-32
local instance
node, 18-20
local I/O, 2-3
local lock, 7-2
LOCAL option
forcing a checkpoint, 18-18, 21-9
verifying access to files, 6-2
lock
boundary, 11-18
conversion, 8-4
cost of, 7-7
dictionary cache, 10-5
DML, 7-3, 10-3
enqueue, 7-3
fine grain, 7-5
global, 18-9
group-based, 8-9, 8-10
hashed, 7-5
identifier, 7-8
implementations, PCM, 7-5

instance, 7-2, 7-4
latch, 7-3
library cache, 10-4
local, 7-2
mastering, 8-9, 19-8
mode compatibility, 9-11
mode, and buffer state, 9-10
mount lock, 7-2, 10-5
name format, 7-8
non-PCM, 7-3, 7-5
OPS exclusive mode, 7-2
OPS shared mode, 7-2
overview of locking mechanisms, 7-1
ownership by IDLM, 8-10
PCM lock, 9-3, 11-15, 17-11
process-owned, 8-10
request, handling by IDLM, 8-3
rollback segment, 6-9
row, 4-15, 10-3
row cache, 6-6
system change number, 10-4
table, 7-3, 7-5, 7-6, 10-3
transaction, 4-13, 7-2, 10-3
types of, 7-2
lock contention
detecting, 19-3
pinpointing, 19-4
lock conversion
detecting, 19-3
excessive rate, 19-7
lock element, 7-9
correspondence to locks, 9-15
creation, 9-19
free, 9-20
LRU list, 9-20
mode, 9-20
name, 9-16
non-fixed mode, 9-20
number, 9-24
valid bit, 9-20
lock operations
in Cache Fusion, 20-6
lock value block, 10-4
log file
accessibility, 5-9
redo log file, 21-1
log history, 14-4, 21-6, 22-9
log sequence number, 21-5, 21-6
log switch, 6-5
adding or dropping files, 14-9
closed thread, 21-11
forcing, 21-10, 21-11, 21-16, A-19
global, 21-16
log history, 21-6
LOG_ALLOCATION parameter (Oracle Version 6), A-19, A-21
LOG_ARCHIVE_DEST parameter, 22-9, 22-10
specifying for recovery, 22-10
LOG_ARCHIVE_FORMAT parameter, 18-10, 21-5, 22-10
same for all instances, 22-10
used in recovery, 22-10
LOG_ARCHIVE_START parameter
automatic archiving, 18-8, 21-3
creating a database, 14-2
LOG_CHECKPOINT_TIMEOUT parameter
inactive instance, 21-10
LOG_DEBUG_MULTI_INSTANCE parameter (Oracle Version 6), A-21
LOG_FILES parameter
obsolete for 8.1, A-5
logical database, 1-18
loosely coupled system
cache consistency, 3-8
characteristics, 3-5
disk access, 3-3
hardware architecture, 3-2, 3-6
tightly coupled nodes, 3-5
LRU list
lock elements, 9-20

M

manual archiving, 21-3, 21-4
dropping a redo log file, 14-9
massively parallel system, 1-5, 11-15
application profile, 3-2
disk access, 3-3
hardware architecture, 3-7
master free list, 11-6

master node, 8-9
mastering, lock, 8-9, 19-8
MAX_COMMIT_PROPAGATION_DELAY
parameter, 4-7, 10-4, 18-10, 18-11
MAX_SORT_SIZE, A-10
MAXDATAFILES option, 14-10
MAXEXTENTS storage parameter
automatic allocations, 11-17, 17-13
preallocating extents, 17-15
MAXINSTANCES option, 11-15, 14-3
changing, 14-10
MAXINSTANCES parameter, 11-13, 23-4
assigning free lists to instances, 11-13, 11-15
calculating non-PCM resources, 16-4
conversion to multi-instance, 23-4, 23-6
MAXLOGFILES option, 14-4, 14-10
MAXLOGHISTORY option, 6-5, 14-4, 21-7
changing, 14-10
CREATE CONTROLFILE, 21-6
log history, 21-6
MAXLOGHISTORY parameter, 23-6
MAXLOGMEMBERS
option, 14-4, 14-10
media failure, 22-1, 22-7, A-22
access to files, 6-2
automatic recovery, 21-7
recovery, A-22
media manager
requirements for creating backups, 21-9
media recovery, 22-7
incomplete, 22-8
log history, 14-4, 21-7, 22-9
O/S utilities, 22-8
member
MAXLOGMEMBERS, 14-4
memory
cache, 4-5
cached data, 4-5
distributed locks, 5-5
IDLM requirements, 8-10
SGA, 6-7
message
access to files, 6-2, 22-4
ALERT file, 6-2, 22-4
distributed lock manager, 9-8
instance shutdown, 18-26
parameter values, 6-1
messaging
in parallel processing, 1-11
migration
data migration, B-1
planning for future, 1-15
returning to exclusive mode, 17-11
MINEXTENTS storage parameter
automatic allocations, 11-17, 17-13, 17-14
default, 17-13
mode
archiving, 4-6, 14-2, 14-9, 21-3
database access, 4-1, 18-12, 18-15
incompatible, 8-2
lock compatibility, 9-11
lock element, 9-20
PCM lock, 4-15
modified data
instance recovery, 22-3
updates, 5-5
modulo, 11-13, 11-15, 17-10, 17-12
MONITOR command
default instance in display screen, 18-19
specifying an instance, 18-21
mount lock, 7-2, 10-5
MOUNT option, 18-13
MPP systems, 1-5
MTS_LISTENER_ADDRESS parameter
obsolete for 8.0.4, A-5
MTS_MULTIPLE_LISTENERS parameter
obsolete for 8.0.4, A-5
multi-instance database
converting application, 23-3
definition, 1-16
reasons not to convert to, 23-2
reasons to convert to, 23-2
requirements, 23-3
multiple shared mode, 4-2, 10-5
multiplexed redo log files, 6-3
example, 6-4
log history, 21-7
total number of files, 14-4
Multi-threaded Server, 8-10
for connection load balancing, 18-32

multiversion read consistency, 4-6

N

Net8
  client-server configuration, 1-20
  connect string, 18-20, 18-21
  distributed database system, 1-18
  for CONNECT, 18-17
  for SET INSTANCE, 18-17
network usage, 19-3
NEXT storage parameter, 6-8, 21-4
NFS, 5-9, 23-4, 23-9
NLS_* parameters, 18-10
NOARCHIVELOG mode, 14-9
  changing mode, 14-2, 14-9, B-2
  creating a database, 14-2, 14-9
  offline backups, 4-6
  requiring offline backups, 21-3
node
  adding, 17-13, 23-2
  affinity awareness, 21-14
  cache coherency, 4-12
  definition, 1-2
  failure, 1-14, 1-19, 22-2
  local, 18-6, 18-8
  parallel backup, 21-12
  parallel shutdown, 18-26
  remote, 18-17, 18-20
  single to cluster, 23-2
NOMOUNT option, 14-3, 22-11
non-fixed mode, lock element, 9-20
non-PCM lock
  dictionary cache lock, 10-5
  DML lock, 10-3
  enqueue, 7-3
  IDLM capacity, 7-6
  library cache lock, 10-4
  mount lock, 10-5
  overview, 10-1
  relative number, 7-6
  system change number, 10-4
  table lock, 10-3
  transaction lock, 10-3
  types, 7-5
  user control, 7-6
non-PCM resources, 16-4
NOORDER option, CREATE SEQUENCE, 6-7
NSTANCE_GROUPS parameter, 18-25
null lock mode, 4-15
number generator, 6-6

O

obsolete parameters, A-20, A-21
offline backup, 4-6, 21-1
  parallel, 21-12
  redo log files, 21-12
offline datafile, A-18
offline tablespace
  deferred rollback segments, A-19
  restrictions, 6-8, A-18
OGMS_HOME parameter, A-5
  obsolete for 8.1, A-5
OLTP applications, 1-3, 1-5, 1-8, 1-12, 1-13, 2-7, 2-8, 3-2, A-2
online backup, 4-6, 21-1
  archiving log files, 21-16
  parallel, 21-12
  procedure, 21-16
  redo log files, 21-12
online datafile
  supported operations, A-18
online recovery, 6-2, 22-2, 22-4, 22-7, A-22
online redo log file
  archive log mode, 14-9
  archiving, 21-1, 21-6
  log switch, 21-6, 21-10
  thread of redo, 6-3, 18-15
online transaction processing, 1-3
OPEN option, 18-13, 18-14
operating system
  exported files, B-1
  Integrated Distributed Lock Manager, 9-8
  privileges, 18-19
  scalability, 2-5
OPS_ADMIN_GROUP parameter, A-6
  obsolete for 8.1, A-5
OPS_FAILOVER clause, A-11
Oracle

background processes, 5-5
backing up, 4-6, 21-1
compatibility, 17-11, B-2
configurations, 1-14
data dictionary, 6-6
datafile compatibility, 6-2
exclusive mode, 4-1, 18-13
executables, 23-8
free space unavailable, 17-11
instance recovery, 22-3
instances on MPP nodes, 3-8
migration, A-17
multi-instance, 7-3
obsolete parameters, A-21
performance features, 4-5
restrictions, 6-7, A-17, A-19
shared mode, 4-2
single-instance, 7-3
version on all nodes, 5-9
Oracle Parallel Server Management(OPSM), A-4
oracle_pid, 11-13
Oradebug, A-4
ORDER option, 6-6, 6-7
overhead
calculating non-PCM resources, 16-4

P

parallel backup, 21-12
parallel cache management, 4-12
parallel cache management lock
acquiring, 4-15
conversion, 9-5
definition, 7-5, 9-2
disowning, 4-15
exclusive, 4-15
how they work, 9-3
implementations, 7-5
minimizing number of, 12-1
null, 4-15
owned by instance LCK processes, 9-5
owned by multiple instances, 9-5
periodicity, 9-8
read, 4-15
relative number, 7-6
releasable hashed, 9-4, 15-7, 15-9
releasing, 4-15
sequence, 4-7
user control, 7-6
parallel database
and parallel execution, 1-21
availability, 1-14
benefits, 1-13
parallel execution
calculating overhead, 16-4
execution processing, 1-2, 1-21, 2-5
limiting instances, 18-27
load balancing, 18-33
processor affinity, A-15
scalability, 12-2
speedup and scaleup, 1-12
under Oracle Parallel Server, 1-21
parallel mode
file operation restrictions, A-17, A-19, B-2
recovery restrictions, A-22
sequence restrictions, 6-7, B-3
shutdown, 18-26
startup, 18-9
PARALLEL option, 18-13
PARALLEL parameter
obsolete for 8.0.4, A-6
parallel processing
benefits, 1-12
characteristics, 1-4
elements of, 1-6
for integrated operations, 1-5
for MPPs, 1-5
for SMPs, 1-5
hardware architecture, 3-2
implementations, 3-2
messaging, 1-11
misconceptions about, 2-12
Oracle configurations, 1-14
types of workload, 1-12
when advantageous, 2-7
when not advantageous, 2-9
parallel processor affinity, A-15
parallel recovery, 22-7, 22-14, 22-15
parallel server, A-4
database configuration, A-4

definition, 1-6
installation, A-4
instance affinity for jobs, A-4
listener load balancing, A-3
Oradebug, A-4
startup and shutdown, 18-14, 18-26
parallel transaction recovery
changes for 8.1, A-3
PARALLEL_DEFAULT_MAX_INSTANCES parameter
obsolete for 8.1, A-5
PARALLEL_DEFAULT_MAXSCANS parameter, 18-11
PARALLEL_INSTANCE_GROUP parameter, 18-24
PARALLEL_MAX_INSTANCES, initialization parameter, 18-27
PARALLEL_MAX_SERVERS parameter, 18-28, 22-15
calculating non-PCM resources, 16-4
ensuring LM lock capacity, 16-6
PARALLEL_SERVER parameter, 18-10, 18-13, 18-14, 18-15
new for 8.0., A-5
PARALLEL_TRANSACTION_RECOVERY parameter
changes for 8.1, A-3
parameter
controlling PCM locks, 9-13
database creation, 11-15, 14-3, 14-4
obsolete, A-21
storage, 6-8, 17-6, 17-8, 17-11
parameter file, 18-3
backing up, 21-1
common file, 18-5, 18-6, 23-7
conversion to multi-instance, 23-6
duplicate values, 18-7
identical parameters, 18-7
NFS access inadvisable, 23-9
PFILE, 18-6, 18-8, 23-5
remote instance, 18-6, 18-8, 18-20
partitioning
application, 12-6
data, 12-7
elements, 2-10
guidelines, 2-10
horizontal, 2-12
vertical, 2-10
partitioning data, 11-12
data files, 6-2, 17-13
free list, 18-16
free lists, 11-2, 11-15, 17-11
index data, 15-5
PCM locks, 11-15, 15-5, 15-6, 17-11
rollback segments, 6-8, 6-9
table data, 11-12, 11-15, 15-5, 17-11
PCM lock
adding datafiles, 15-12
allocating, 15-2
calculating, 15-17
checking for valid number, 15-11, 15-13
contention, 11-15, 15-5, 15-6, 17-11
conversion time, 15-16
estimating number needed, 15-3
exclusive, 9-25
fixed fine grain, 9-4
fixed hashed, 9-3
index data, 15-5
mapping blocks to, 9-6, 11-15, 17-11
planning, 15-2
releasable fine grain, 9-3
releasable hashed, 9-4
resources, 15-17
sessions waiting, 15-17
set of files, 9-7
shared, 9-25, 15-5
specifying total number, 14-10
valid lock assignments, 15-11
worksheets, 15-4
PCTFREE, 11-5, 19-6
PCTINCREASE parameter
table extents, 17-11
PCTUSED, 11-5
performance
and lock mastering, 8-9
application, 11-12
benefits of parallel database, 1-13
caching sequences, 6-7
fine grain locking, 9-20
inserts and updates, 11-12

Oracle8 features, 4-5
rollback segments, 6-8, 6-9
sequence numbers, 6-7
shared resource system, 1-14
persistent resource, 8-10
PFILE option, 18-6, 18-8
conversion to multi-instance, 23-5
pinging, 9-8, 9-9, 15-14, 15-16
definition, 9-2
false, 9-17
rate, 15-15
ping/write ratio, 19-4
PL/SQL shared memory area, 5-5
PMON process, 5-5
pre-allocating extent, 11-17
prime number, A-7
private rollback segment
acquisition, 6-8
creating, 14-5
individual parameter file, 18-5
specifying, 6-9
private thread, 14-8
privilege
ALTER SYSTEM, 21-10
process free list
definition, 11-5
pinging of segment header, 11-6
PROCESSES parameter, 18-10
calculating non-PCM resources, 16-4
ensuring LM lock capacity, 16-6
processor affinity
parallel execution, A-15
protected write mode, 8-7
public rollback segment
bringing online, 14-6
common parameter file, 18-5
creating, 14-6
owner, 14-6
specifying, 14-6
using by default, 14-6
PUBLIC thread, 14-8

R

random access, 2-8
raw device, 23-4
read consistency
multiversion, 4-6
rollback information, 6-8
read lock mode, 4-15
reader/writer conflicts
and Cache Fusion, 20-1
read-only access, 4-6, 4-15
applications, 2-7
index data, 15-5
read PCM lock, 4-15
read-only tables, 12-2
RECO process, 1-18, 5-5
RECOVER command, 18-19, 22-7, 22-13, 22-15
RECOVER DATABASE statement, 22-7, 22-8
RECOVER DATAFILE statement, 22-7, 22-8
RECOVER TABLESPACE statement, 22-7, 22-8
recovery, 22-1
access to files, 6-2, 22-4
after SHUTDOWN ABORT, 18-27
archive history, 14-4
automatic, 21-7
conversion to multi-instance, 23-9
deferred transaction, A-12
definition, 22-1
detection of error, 19-8
disaster, 22-10, 22-13
FREEZE_DB_FOR_INSTANCE_RECOVERY, 2
2-4
from an offline backup, 22-10
from an online backup, 22-10
from multiple node failure, 22-3
from single-node failure, 22-2
global checkpoint, 21-9
incomplete media, 22-8
instance, 14-4, 18-27, 22-2
instance failure, 22-1
instance recovery, 22-1
log history, 21-7, 22-9
media failure, 6-2, 21-7, 21-10, 22-6, 22-7, A-22
online, 22-2
parallel, 22-14, 22-15
PARALLEL_MAX_SERVERS parameter, 22-15
phases, 22-5
Recovery Manager, 22-8

recovery time, 21-10
restrictions, A-22
rolling back, 6-8
setting parallelism, 22-14, 22-15
starting another instance, 14-4
using redo log, 21-12
Recovery Manager, 22-6
archive log backup, 21-7
disaster recovery, 22-10
incomplete media recovery, 22-8
media recovery, 22-8
RECOVERY_PARALLELISM parameter, 18-10, 22-1, 22-14, 22-15
redo log
archiving mode, 21-3
instance recovery, 22-2
log history, 21-6
reconfiguring, 14-9
redo log buffer, 5-5
redo log file
accessibility, 5-9
adding, A-18
archiving, 4-6, 14-9, 21-1, 21-3, 21-10
backup, 21-12
dropping, 21-10, A-18
identified in control file, 6-5
log history, 21-6
log sequence number, 21-5
multiplexed, 21-7
overwriting, 4-6, 21-3
renaming, 21-10, A-18
thread of redo, 6-3
redo thread, 21-4
registration
of instances, 18-28
relative file number, 6-3
releasable freelist waits, 15-13
releasable hashed PCM lock, 9-4, 15-7, 15-9
remote databases, 1-18
remote instance, 18-6, 18-8, 18-20
remote I/O, 2-3
REMOTE_LOGIN_PASSWORDFILE parameter, 18-26
renaming a file
log switch, 21-10
redo log file, A-18
RENAME FILE option, A-18
replicated systems, 23-1
resource
database, 4-12
operating system, 18-9
persistent, 8-10
releasing, 22-3
response time, 1-10
RESTORE DATABASE statement, 22-8
RESTORE DATAFILE statement, 22-8
RESTORE TABLESPACE statement, 22-8
restrictions
cached sequence, 6-7
changing the redo log, 14-9
deferred rollback segments, A-19
file operations, A-17, A-19, B-2
offline tablespace, 6-8, A-18
recovery, A-22
RETRY option, 18-15
STARTUP PARALLEL command, 18-15
RMAN
ALLOCATE CHANNEL command, 21-8
rollback segment, 14-5
contention, 6-8, 6-9, 14-5
deferred, 6-9, A-19
description, 6-8
distributed locks, 6-9
global constant parameters, 6-9
ID number, 14-5, 14-7
multiple, 6-8, 14-5, 18-15
name, 14-5, 14-7
online, 6-8, 14-7
onlining, A-13
private, 23-9
public, 14-6
public vs. private, 6-9, 14-6, 18-5
specifying, 14-5
SYSTEM, 6-8
tablespace, 6-8, 14-5, 14-7, 23-9
ROLLBACK_SEGMENTS parameter, 6-9, 6-10, 18-10, 18-11
conversion to multi-instance, 23-7
private and public segments, 14-5, 14-6
rolling back

instance recovery, 22-3
rollback segments, 6-8
row locks, 4-15
routing, data-dependent, 12-7, 19-6
row cache, 6-6
row level locking, 7-2
DML locks, 10-3
independent of PCM locks, 4-15
resource sharing system, 4-6, 5-4
ROW_CACHE_MULTI_INSTANCE parameter (Oracle Version 6), A-21
ROW_LOCKING parameter, 18-11

S

scalability
application, 2-6, 2-13
applications, 2-2
database, 2-5
definition, 1-8
determinants, 1-13
disk input and output, 2-3
enhancement for release 7.3, A-11
four levels of, 2-1
hardware, 2-2
network, 2-6
operating system, 2-5
potential, 1-12
relative, 2-8
shared memory system, 2-5
SCN, 4-4
System Change Number, 10-4
SCSI, 3-3
segment
definition, 11-3
header, 7-9
header block, 11-14, 14-7
header contention, 11-6, 19-6
header, contention, 11-6
ID number, 14-5, 14-7
name, 14-7
rollback segment, 6-8
size, 14-7
sequence
data dictionary cache, 4-7, 6-7
log sequence number, 21-5, 21-6
not cached, 6-7, B-3
timestamp, 6-7
sequence number generator
application scalability, 2-6
contention, 2-9
distributed locks, 6-6
LM locks, 4-7
on parallel server, 6-6
restriction, 6-7, B-3
skipping sequence numbers, 6-7
SEQUENCE_CACHE_ENTRIES parameter
obsolete for 8.1, A-5
sequential processing, 1-2, 1-4
SERIALIZABLE parameter, 18-11
Server Manager
privileged commands, 18-19
service connections
to clients, 18-28
SERVICE_NAMES parameter
in instance registration, 18-31
session
multiple, 18-7, 18-21, 18-27
waiting for PCM lock conversion, 15-17
SESSIONS parameter
ensuring LM lock capacity, 16-6
SET INSTANCE command, 18-6, 18-18, 18-20
instance startup, 18-6, 18-20
requires Net8, 18-17
SET UNTIL command, 22-11
shared disk system
advantages, 3-6
implementations, 3-3
scalability, 2-3
with shared nothing system, 3-9
shared exclusive mode, 8-7
shared memory system
scalability, 2-5
tightly coupled, 3-3
shared mode
database access, 4-2
datafiles, 6-2
file operation restrictions, A-18
instance number, 18-16
instance recovery, 22-2

recovery restrictions, 22-7
startup, 18-15
shared nothing system
advantages, 3-8
disadvantages, 3-8
disk access, 3-2
massively parallel systems, 3-7
overview, 3-7
scalability, 2-3
with shared disk system, 3-9
SHARED option, 18-13
shared resource system, 17-13
shared SQL area, 5-5, 12-6
SHOW INSTANCE command, 18-19, 18-20
SHOW PARAMETERS command, 18-19, 18-21
instance number, 18-16
SHOW SGA command, 18-19, 18-21
SHUTDOWN command
ABORT option, 18-27
IMMEDIATE option, 18-27
specifying an instance, 18-20
shutting down an instance, 18-26
abnormal shutdown, 18-27
archiving redo log files, 21-10
changing startup order, 18-16
forcing a log switch, 21-10
lost sequence numbers, 6-7
unarchived log files, 21-4
SID parameter, 18-6
single instance database, 1-15
single shared mode, 4-2, 10-5
SINGLE_PROCESS parameter, 18-15
SIZE option
allocating extents, 17-14
SMON process, 5-6
instance recovery, 22-2, 22-3
recovery after SHUTDOWN ABORT, 18-27
transaction recovery, A-13
SMP, 1-15
sort enhancements, A-10
SORT MERGE JOIN, 12-3
sort space, A-10
SORT_DIRECT_WRITES parameter, A-14
space
allocating extents, 17-13
deallocating unused, 17-16
determining unused, 17-16
free blocks, 11-2, 11-17
free list, 11-2
not allocated to instance, 11-6, 17-12
SGA, 18-21
sources of free blocks, 11-6
unavailable in exclusive mode, 17-11
specialized servers, 1-5
speed-down, 1-8, 1-13
speedup
definition, 1-7
with batch processing, 1-13
SQL area
shared, 12-6
SQL statement
instance-specific, 18-18
restrictions, B-2
standby databases, 18-33, 22-16
starting up
after file operations, 15-6, A-18
creating database and, 14-3
during instance recovery, 14-4
exclusive mode, 17-13, 18-12
global constant parameters, 6-9, 18-9
remote instance, 18-6, 18-7, 18-8, 18-20
rollback segments, 6-8, 14-5
shared mode, 18-15, A-18
startup order, 18-16
verifying access to files, 6-2
STARTUP command, 14-3, 18-6, 18-13
MOUNT option, 22-13
OPEN option, 18-13
PFILE option, 18-6, 18-8
specifying an instance, 18-20
static hashing
lock mastering scheme, 8-9
statistics
tuning, 19-2
storage options
clustered tables, 17-6, A-20
extent size, 6-8, 17-11, 17-13, 17-14
index, 17-8
rollback segment, 6-8
table, 17-6

stored procedures, 7-8
sub-shared exclusive mode, 8-7
sub-shared mode, 8-7
switch archiving mode, 14-2, 14-9, B-2
symmetric multiprocessor, 2-5, 3-2, 3-4
  configuration, 1-15
  in loosely coupled system, 3-5
  parallel processing, 1-5
synchronization
  cost of, 1-10, 1-17, 2-9, 2-11, 2-12
  non-PCM, 4-16
  overhead, 1-10
SYSDBA, 18-13, 18-21, 18-26, 21-11
SYSOPER, 18-13, 18-21, 18-26, 21-11
system change number (SCN), 10-4
  archive file format, 21-5
  archiving redo log files, 21-4
  incrementation, 4-4
  Lamport, 4-7
  non-PCM lock, 7-5
  redo log history, 21-6
System Global Area (SGA)
  in parallel server, 5-5
  instance, 5-4
  parameter file, 18-3
  row cache, 6-6
  sequence cache, 6-6
  SHOW SGA command, 18-21
  statistics, 18-21
SYSTEM rollback segment, 6-8
SYSTEM tablespace, 14-5
system-specific Oracle documentation
  archived redo log name, 21-5
  client-server processing, 1-20
  datafiles, maximum number, B-3
  free list overhead, 11-5
  instance number range, 17-12
  MAXLOGHISTORY default, 21-6
  Net8 connect string, 18-17
  password file name, 18-26
  recovery process allocation, 22-15
  redo log archive destination, 21-6
  redo log archive format, 21-6
  supported Oracle configurations, 1-14
  system change number
    Lamport, 4-7
    SCN, 4-5

T

table
  access pattern, 12-2
  allocating extents, 11-11, 17-14
  cluster, 17-7
  contention, 6-8, 17-13
  free space unavailable, 17-11
  initial storage, 11-17, 17-13
  lock, 7-3, 7-6
  multiple files, 11-12, 17-13
  PCM locks, 11-15, 17-11
  read-only, 12-2
  tablespace, 6-8
table lock, 10-3
  disabling, 16-7
  minimizing, 16-6
TABLE_LOCK column, 16-7
tablespace
  active rollback segments, 6-8
  backup, 4-6, 21-1
  creating, 15-6, A-18
  data files, A-18
  dropping, 15-6, A-18
  index data, 15-5
  offline, 6-8
  online rollback segments, 14-5, 14-7
  parallel backup, 21-12
  parallel recovery, 22-7
  read-only, 15-11
  recovery, 22-7, A-22
  rollback segment, 6-8, 14-5, 14-7
  SYSTEM, 14-5
  tables, 6-8
  taking offline, 6-8, A-18, A-19
thread
  archive file format, 21-5
  archiving redo log files, 21-4, 21-10
  associated with an instance, 14-8
  closed, 21-16
  disabled, 14-9
  enabled, 21-7, 21-16, 22-10

example, 6-3
exclusive mode, 18-15
forced log switch, 21-10
log history, 21-7
number of groups, 6-4
open, 21-7, 21-16
public, 14-8
single, 4-8
THREAD option, 18-18, 21-4, 21-10
creating private thread, 6-3
creating public thread, 6-3
disabling a thread, 14-9
when required, 14-8
THREAD parameter, 14-8, 18-5, 18-10
conversion to multi-instance, 23-7
individual parameter files, 18-5
instance acquiring thread, 6-3
tightly coupled system
hardware architecture, 3-3, 3-5
implementations, 3-2
in loosely coupled cluster, 3-5
performance, 3-4
TM, DML Enqueue, 10-3
TNSNAMES.ORA
configuring for instance registration, 18-29
TP monitor, A-11
trace file
conversion to multi-instance, 23-4
transaction
aborted, 6-8
committed data, 4-6, 21-9
concurrent, 4-6, 4-12, 5-4
free list, 11-4
inserts, 4-8, 11-2
instance failure, 22-3
isolation, 4-15
lock, 4-13, 7-2, 7-3, 7-5, 7-6, 10-3
offline tablespace, 6-9, A-19
recovery, A-12
rollback segments, 6-9, A-19
rolling back, 6-8, 22-3
row locking, 4-6, 4-15
sequence numbers, 6-6
updates, 4-6, 11-2
waiting for recovery, 22-3
transaction processing monitor, 12-7
TRANSACTIONS parameter, 6-10
calculating non-PCM resources, 16-4
ensuring LM lock capacity, 16-6
TRANSACTIONS_PER_ROLLBACK
parameter, 6-10
tuning, 19-2
two-phase commits, 1-19
TX, Transaction, 10-3

U

updates
at different times, 2-7
concurrent, 4-6, 11-14
free lists, 11-14, 18-16
instance lock, 9-8
PCM lock, 4-15
performance, 11-12
upgrade
Oracle, 23-1
replicated systems, 23-1
user
benefits of parallel database, 1-14
multiple, 5-4
name and password, 18-20
PUBLIC, 14-6, 14-7
SYS, 14-7
user process
free list, 11-2, 11-15, 17-8
instance shutdown errors, 18-26
manual archiving, 21-4
USER_TABLES table, 16-7
user-level IDLM, 8-10
user-mode IPCs
and Cache Fusion, 20-6
utilities, Oracle
Export, Import, B-1

V

V$ACTIVE_INSTANCES view, 23-4, A-9
V$BH view, 9-10, 19-5, A-7, A-9, A-17
V$CACHE view, 19-5, A-17
V$CACHE_LOCK view, 20-12

V$CLASS_PING view, A-7
V$DATAFILE view, 6-3, 15-11
V$DLM_ALL_LOCKS
  new view for 8.1, A-2
V$DLM_ALL_LOCKS view, 7-5, 8-11
V$DLM_CONVERT_LOCAL view, 8-11, A-7
V$DLM_CONVERT_REMOTE view, 8-11, A-7
V$DLM_LATCH view, A-7
V$DLM_LOCKS view, 7-5, 8-11
  changed for 8.0.4, A-6
V$DLM_MISC view, 8-11, A-7
V$DLM_RESS
  new view, A-2
V$DLM_RESS view, 7-5, 8-11
V$FAST_START_SERVERS
  view, 22-16, A-3
V$FAST_START_TRANSACTIONS
  view, 22-16, A-3
V$FILE_PING view, A-7
V$LE table, 9-20
V$LOCK view, 7-9
V$LOCK_ACTIVITY view, 20-12, A-17
  detecting lock conversion, 19-3
V$LOCK_ELEMENT view, 7-9, 9-20
V$LOCKS_WITH_COLLISIONS view, 20-12, A-16
V$LOG_HISTORY view, 21-7
V$LOGFILE view, 6-5
V$PING view, 19-5, 19-6, A-17
V$RECOVERY_LOG view, 21-7
V$RESOURCE_LIMIT view, 16-3, A-7
V$ROLLNAME view, 20-12
V$SESSION_WAIT view, 15-17
V$SORT_SEGMENT view, A-9, A-10
V$SYSSTAT view, 15-13, 22-5, A-7
  detecting lock conversion, 19-3
V$SYSTEM_EVENT view, 15-16
V$THREAD view, 23-4
V$WAITSTAT view, 19-6
valid bit, lock element, 9-20
version compatibility
  on the same cluster, 18-22
versions, Oracle
  compatibility, 17-11, A-17
  upgrading, A-1
vertical partitioning, 2-10
VIA
  interconnect protocol, 20-6
view
  global, 18-25, 20-12
  rollback segments, 14-6
virtual memory usage, 19-3

W

wait time, 1-4, 1-10
wait, session, 15-17
workloads
  and scaleup, 1-13
  balancing, 1-6
  mixed, 1-5
  partitioning, 1-21
  type of, 1-5, 1-12

X

XA interface
  library, 8-10
  performance enhancement, A-11
  recovery enhancement, A-11
XA_RECOVER call, A-11