Oracle8*i* Parallel Server

Administration, Deployment, and Performance

Release 2 (8.1.6)

December 1999

Part No. A76970-01

Oracle8*i* Parallel Server Administration, Deployment, and Performance, Release 2 (8.1.6)

Part No. A76970-01

Copyright © 1996, 1999, Oracle Corporation. All rights reserved.

Primary Author: Mark Bauer.

Contributing Author: Cathy Baird.

Primary Contributors: David Austin, Wilson Chan, and Michael Zoll.

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

Graphic Designer: Valarie Moore.

The Programs (which include both the software and documentation) contain proprietary information of Oracle Corporation; they are provided under a license agreement containing restrictions on use and disclosure and are also protected by copyright, patent, and other intellectual and industrial property laws. Reverse engineering, disassembly, or decompilation of the Programs is prohibited.

The information contained in this document is subject to change without notice. If you find any problems in the documentation, please report them to us in writing. Oracle Corporation does not warrant that this document is error free. Except as may be expressly permitted in your license agreement for these Programs, no part of these Programs may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Oracle Corporation.

If the Programs are delivered to the U.S. Government or anyone licensing or using the programs on behalf of the U.S. Government, the following notice is applicable:

Restricted Rights Notice Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

The Programs are not intended for use in any nuclear, aviation, mass transit, medical, or other inherently dangerous applications. It shall be the licensee's responsibility to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of such applications if the Programs are used for such purposes, and Oracle Corporation disclaims liability for any damages caused by such use of the Programs.

Oracle, SQL*Loader, Secure Network Services, and SQL*Plus are registered trademarks of Oracle Corporation, Redwood Shores, California. Oracle Call Interface, Oracle8*i*, Oracle8, Oracle Parallel Server, Oracle Forms, Oracle TRACE, Oracle Expert, Oracle Enterprise Manager, Oracle Server Manager, Net8, PL/SQL, and Pro*C are trademarks of Oracle Corporation, Redwood Shores, California.

Contents

Part VI Oracle Parallel Server Reference

Send Us Your Comments

***Oracle8i Parallel Server Administration, Deployment, and Performance*, Release 2 (8.1.6)**

Part No. A76970-01

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

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

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

- Electronic mail: infodev@us.oracle.com
- FAX: (650) 506-7228 Attn: ST/Oracle8*i* Documentation Manager
- Postal service:
  Oracle Corporation
  ST/ORacle8*i* Documentation Manager
  500 Oracle Parkway, 4OP12
  Redwood Shores, CA 94065
  USA

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

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

Preface

This manual describes the administrative and deployment tasks for the Oracle Parallel Server. You should read this book after you have completed the procedures in the *Oracle8i Parallel Server Setup and Configuration Guide*. You should also have read *Oracle8i Parallel Server Concepts*.

This manual prepares you to implement parallel processing by describing the administrative procedures to follow after installing Oracle Parallel Server. It also advises you how to develop and deploy applications, as well as how to tune them. Information in this manual applies to Oracle Parallel Server as it runs on all operating systems. Where necessary, this manual refers to platform-specific documentation.

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

What's New in Oracle8*i*?

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

> **See Also:** *Oracle8i Parallel Server Concepts* for information on feature changes from one release of Oracle Parallel Server to another.

Release 8.1.5

Release 8.1.5 introduced the first phase of cache fusion.

Release 8.1.6

Release 2 (8.1.6) introduces further enhancements to cache fusion as well as the Primary / Secondary instance feature. There are also several new performance statistics.

Intended Audience

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

How this Book is Organized

This book presents Oracle Parallel Server administration, deployment, and performance in five parts. It begins with the fundamental administration of Oracle Parallel Server and then presents application and database design and deployment for Parallel Server. The last parts of the book present performance tuning as well as maintenance topics, such as backup and recovery.

Structure

The following describes the five parts of this book.

Part I, "Administering Oracle Parallel Server Parameter Files"

Chapter 1, "Parameter Files and Oracle Parallel Server-Specific Parameters"	This chapter describes the parameter files and Oracle Parallel Server-specific parameters.

Part II, "Oracle Parallel Server Administration"

Chapter 2, "Parallel Execution in Oracle Parallel Server Environments"	This chapter describes parallel execution as used within Oracle Parallel Server environments.
Chapter 3, "Oracle Parallel Server Database Creation Issues"	This chapter describes Oracle Parallel Server database creation issues. The information in this chapter is supplemental to the information in the *Oracle8i Parallel Server Setup and Configuration Guide*.
Chapter 4, "Administering Instances"	This chapter describes the basic administrative procedures for instance management.

Part III, "Oracle Parallel Server Design and Deployment"

Chapter 5, "Application Analysis and Partitioning"	This chapter describes analysis and partitioning for Oracle Parallel Server environments.
Chapter 6, "Database Design Techniques"	This chapter describes Oracle Parallel Server database design issues such as block and extent operations, contention reduction, and locking strategies.
Chapter 7, "Planning the Use of PCM and Non-PCM Instance Locks"	This chapter explains how to use and maintain both PCM and non-PCM locking resources.
Chapter 8, "Using Free List Groups to Partition Data"	This chapter explains how to improve performance by using free list groups to partition data.
Chapter 9, "Setting Instance Locks"	This chapter describes how to set instance locks within Oracle Parallel Server environments.
Chapter 10, "Ensuring DLM Capacity for Locks and Resources"	This chapter explains how to plan and manage the use of locks and resources to optimize performance.

Part IV, "Oracle Parallel Server Performance Monitoring and Tuning"

Chapter 11, "General Tuning Recommendations"	This chapter presents general tuning recommendations.
Chapter 12, "Tuning Oracle Parallel Server and Inter-Instance Performance"	This chapter describes how to monitor and tune inter-instance performance issues.

Part V, "Oracle Parallel Server Maintenance"

Chapter 13, "Backing Up Your Database"	This chapter describes the procedures for backing up Oracle Parallel Server databases.
Chapter 14, "Recovering the Database"	This chapter describes the procedures for recovering Oracle Parallel Server databases.

Related Documents

Before reading this manual, you should read *Oracle8i Parallel Server Concepts* and the *Oracle8i Parallel Server Setup and Configuration Guide.*

You can also read the following manuals for more information:

Installation Guides

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

Operating System-Specific Administrative Guides

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

Oracle Parallel Server Management

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

Oracle Server Documentation

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

Conventions

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

- Text
- Syntax diagrams and notation
- Code examples

Text

This section explains the conventions used within the text:

UPPERCASE Characters

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

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

Italicized Characters

Italicized words within text are book titles or emphasized words.

Syntax Diagrams and Notation

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

Keywords

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

Parameters

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

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

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

subquery	The substitution value must be a SELECT statement contained in another SQL statement.	SELECT ename
		FROM emp

statement_name *block_name*	The substitution value must be an identifier for a SQL statement or PL/SQL block.	s1 b1

Code Examples

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

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

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

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

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

Part I

Administering Oracle Parallel Server Parameter Files

Part One explains the administrative issues regarding initialization parameter files. This part also describes Oracle Parallel Server-specific parameter issues that you must consider after configuring your Oracle Parallel Server environment. This part includes the following chapter:

1 Parameter Files and Oracle Parallel Server-Specific Parameters

This chapter describes the initialization parameter files and Oracle Parallel Server-specific parameters. It includes the following sections:

- Managing Parameter Files for Oracle Parallel Server
- Setting Initialization Parameters for Multiple Instances

Chapter 2 discusses additional parameters for parallel execution in Oracle Parallel Server environments.

Managing Parameter Files for Oracle Parallel Server

In addition to the parameters used in single instance environments, there are several Oracle Parallel Server-specific parameters. Some of these parameters must have identical values across all instances.

You can specify settings for these parameters using one or more parameter files that you edit with any text editor. Oracle reads these settings from the parameter files and writes the values to the control files.

You can implement parameter files in Oracle Parallel Server in several ways using:

- One Common Parameter File
- Instance-Specific Parameter Files
- Non-Default Parameter Files For Particular Sessions

Parameter File Naming Conventions

Oracle recommends using these naming conventions for parameter files:

- Name the common parameter file `init_`*`dbname`*`.ora` where `DBNAME` is the name of the Oracle Parallel Server database.
- Name any instance-specific parameter files `init_`*`sid`*`.ora`, where `SID` is the instance name or number.
- Name any non-default parameter files.

Using these naming conventions simplifies Oracle Parallel Server administration.

One Common Parameter File

If you use all the default parameter settings with Oracle Parallel Server, place one common parameter file on the shared disk. This simplifies administration because you use one file to globally manage parameter settings. If your clustering system does not share files, copy the common file onto each node.

> **See Also:** *Oracle8i Parallel Server Setup and Configuration Guide* for details about `init_`*`dbname`*`.ora` initialization parameter file entries.

Instance-Specific Parameter Files

For some configurations, you may want to use instance-specific parameter settings to improve performance. For example, you can create System Global Areas (SGAs) of different sizes for each instance. If you do this on a shared file system, also use the common parameter file for parameters that must have identical settings across all instances. Oracle recommends that you identify the common file from within the instance-specific parameter file by setting the IFILE (include file) parameter.

Note: The Database Configuration Assistant does this by default.

Conditions Under Which You Must Use Instance-Specific Files

You must use instance-specific parameter files when you create instances that:

- Specify INSTANCE_NUMBER
- Specify THREAD
- Use private rollback segments

Instances that use only public rollback segments can share a common parameter file.

See Also: "Parameters for Common Parameter Files" on page 1-9.

Placement and Use of IFILE Parameters within Instance-Specific Files

If you duplicate parameter entries in a parameter file, the last value specified in the file for the parameter overrides earlier values. To ensure Oracle uses the correct common parameter values, place the IFILE parameter at the end of any instance-specific parameter files. Conversely, you can override common parameter values by placing the IFILE parameter before the instance-specific parameter setting.

Note: The Database Configuration Assistant places the IFILE parameter at the top of the parameter file.

Using Multiple IFILEs

You can specify IFILE more than once in a parameter file to include multiple common parameter files. If you do not reset a parameter value in each subsequent common parameter file designated with your multiple IFILE entries, then each

IFILE does not override previous values. For example, an instance-specific parameter file might include an `init_dbname.ora` file and separate parameter files for the LOG_* and GC_* parameters as in this example:

```
IFILE=INIT_OPS.ORA
IFILE=INIT_LOG.ORA
IFILE=INIT_GC.ORA
LOG_ARCHIVE_START=FALSE
THREAD=3
ROLLBACK_SEGMENTS=(RB_C1,RB_C2,RB_C3)
```

In this example, the value of LOG_ARCHIVE_START overrides any value specified in `init_log.ora` for this parameter because the IFILE parameter appears before the LOG_ARCHIVE_START parameter.

See Also:

- "Parameters That Must Be Identical Across All Instances" on page 1-8.
- "Shutting Down Instances" on page 4-5.

Non-Default Parameter Files For Particular Sessions

Specify a non-default parameter file for a particular session by using the PFILE option of the STARTUP command. Do this, for example, to use special parameter settings to optimize parallel execution for off-peak, batch operations.

The parameter file you specify with PFILE must be on a disk accessible to the local node, even if you specify a parameter file for an instance on a remote node. You can have multiple non-default parameter files and use them on demand.

Location of Initialization Files

The database for which the instance is started must have access to the initialization parameter files. Oracle Parallel Server uses the database initialization files located in:

- `ORACLE_HOME\admin\db_name\pfile` on Windows NT
- `$ORACLE_HOME/admin/db_name/pfile` on UNIX

The Startup Process and Parameters in Parallel Server Environments

As mentioned, Oracle writes the parameter values from all parameter files to the control files when the first instance in your environment starts up. The alert log of the first instance identifies that instance as the first one to start and mounts the database. The startup process is described in more detail in Chapter 4.

If the parameter file for a subsequent instance contains a parameter that must be the same for all instances and its value does not match the value already set for that parameter in the control file, the instance cannot mount the database.

Note: To locate your alert log file, use the search string `alert*.log`.

Starting Two Instances on Remote Nodes

You can start multiple nodes from a SQL*Plus session on one node. For example, you can use a SQL*Plus session on a local node to start two instances on remote nodes using individual parameter files named `init_`*`ops1`*`.ora` and `init_`*`ops2`*`.ora`:

Before connecting to the database, in SQL*Plus direct your commands to the first instance by entering:

```
SET INSTANCE OPS1;
```

Connect to the first instance, start it, and disconnect from it by entering:

```
CONNECT INTERNAL;
STARTUP PFILE=INIT_OPS1.ORA;
DISCONNECT;
```

Redirect commands to the second instance:

```
SET INSTANCE OPS2;
```

Connect to and start the second instance by entering:

```
CONNECT INTERNAL;
STARTUP PFILE=INIT_OPS2.ORA;
```

Here, OPS1 and OPS2 are Net8 net service names for the two instances. These net service names are defined in `TNSNAMES.ORA`.

Both individual parameter files can use the IFILE parameter to include parameter values from the `init_`*`dbname`*`.ora` file.

Instance Numbers and Startup Sequence

You can explicitly specify an instance number by using the initialization parameter INSTANCE_NUMBER when you start it with Oracle Parallel Server enabled or disabled. You should set INSTANCE_NUMBER equal to the value of THREAD_ID. If you do not specify an instance number, the instance automatically acquires the lowest available number.

Always use the INSTANCE_NUMBER parameter if you need consistent allocation of extents to instances for inserts and updates. This allows you to maintain data partitioning among instances. You must specify unique instance numbers for each instance when using the INSTANCE_NUMBER parameter.

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.

Startup Order Determines Instance Number by Default

The startup order determines the instance number if you have not specified a value for INSTANCE_NUMBER. Note that default startup numbers are difficult to control if instances start up in parallel. Moreover, instance numbers can change after you shut down and restart instances. The SQL*Plus command:

```
SHOW PARAMETER INSTANCE_NUMBER
```

Shows the current number for each instance. This command displays a null value if Oracle assigned an instance number based on startup order.

After you shut down an instance, its instance number is available for re-use. 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 another method. All instance numbers must be unique, regardless of the method by which they are acquired.

An instance starting with Oracle Parallel Server 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 merely to perform administrative operations with Oracle Parallel Server disabled, you can omit the INSTANCE_NUMBER parameter from the parameter file.

An instance starting with Oracle Parallel Server disabled can also specify a thread other than 1 to use the online redo log files associated with that thread.

See Also:

- "Creating Additional Rollback Segments" on page 3-4.
- Chapter 8 for information about allocating free space for inserts and updates.
- *Oracle8i Administrator's Guide* for more information on starting up Oracle databases.

Setting Initialization Parameters for Multiple Instances

This section discusses Oracle Parallel Server initialization parameters for multiple instances and covers the following topics:

- Parameters That Must Be Identical Across All Instances
- Parameters That Must Be Unique Across All Instances
- Parameters for Common Parameter Files
- GC_* Global Cache Parameters
- Parameter Notes for Multiple Instances

See Also: *Oracle8i Reference* for details about other Oracle initialization parameters.

Parameters That Must Be Identical Across 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 Oracle Parallel Server. Specify these parameter values in the common parameter file, or within each `init_`*`dbname`*`.ora` file on each instance. Table 1–1 lists the parameters that must be identical on every instance.

Table 1–1 Parameters That Must Be Identical on All Instances

CONTROL_FILES	LM_LOCKS and LM_RESS (automatically calculated by Oracle, but identical values recommended)
DB_BLOCK_SIZE	LOG_ARCHIVE_DEST (optional)
DB_FILES	MAX_COMMIT_PROPAGATION_DELAY
DB_NAME	SERVICE_NAMES
DB_DOMAIN	ACTIVE_INSTANCE_COUNT
DML_LOCKS	ROW_LOCKING
GC_FILES_TO_LOCKS	GC_ROLLBACK_LOCKS
PARALLEL_SERVER_INSTANCES	DML_LOCKS (only if set to zero)

Parameters That Must Be Unique Across All Instances

If you use the parameters INSTANCE_NUMBER, THREAD, or ROLLBACK_SEGMENTS, Oracle recommends setting unique values for them using instance-specific parameter files.

- Oracle uses the INSTANCE_NUMBER parameter to distinguish among instances at startup. Oracle also uses INSTANCE_NUMBER to assign free space to instances using the INSTANCE option of the ALLOCATE EXTENT clause in the ALTER TABLE or ALTER CLUSTER statements. If you assign one instance a value for INSTANCE_NUMBER, you must do so for all instances and use unique values for each instance.
- Specify the THREAD parameter so instances avoid the overhead of acquiring different thread numbers during startup and shutdown. Oracle uses the THREAD number to assign redo log file groups to specific instances. To simplify administration and avoid confusion, use the same number for the THREAD parameter that you used for the INSTANCE_NUMBER parameter.
- Private rollback segments can improve performance on some systems because they create less write contention than public rollback segments. Oracle acquires

private rollback segments upon instance startup based on the files you identify with the ROLLBACK_SEGMENTS initialization parameter. If you do not declare a rollback segment filename with this parameter for an instance, Oracle acquires public rollback segments for the instance.

Parameters for Common Parameter Files

This section on common parameter files includes the following topics:

- DB_NAME Parameter
- GC_* Global Cache Parameters
- Multiple Instance Issues for Initialization Parameters
- LM_* Initialization Parameters

DB_NAME Parameter

Make sure you include the DB_NAME parameter in the common parameter file. If you do not set a value for DB_NAME in the common file, you must set a value for this parameter in the instance or non-default parameter files. The value you set for this parameter must be identical for all instances.

If you specify parameters with identical values in a common parameter file referred to by IFILE, you can omit parameters for which you are using the default values.

GC_* Global Cache Parameters

Initialization parameters with the prefix GC (Global Cache) are relevant only to Oracle Parallel Server. 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 also affect the use of certain operating system resources. Specify these parameters in the `init_`*`dbname`*`.ora` file.

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 the correct values for its global cache parameters.

The global cache parameters for Oracle Parallel Server are:

Parameter:	Description:
GC_FILES_TO_LOCKS	Controls the ratio of data block locks to data blocks. (must be set identically on all instances).
GC_ROLLBACK_LOCKS	Controls the number of undo block locks. (must be set identically on all instances).
GC_RELEASABLE_LOCKS	Controls the number of releasable locks.
GC_DEFER_TIME	Specifies the amount of time in hundredths of seconds that Oracle waits before responding to forced-write requests from other instances for hot blocks.

See Also: Part Three of this book, "Oracle Parallel Server Design and Deployment" for a thorough discussion of setting global cache parameters.

Multiple Instance Issues for Initialization Parameters

Table 1–2 summarizes multi-instance issues concerning initialization parameters.

Table 1–2 Initialization Parameter Notes for Multiple Instances

Parameter	Description and Comments
DML_LOCKS	Must be identical on all instances only if set to zero. The value should equal the total of locks on tables currently referenced by all users. For example, if three users are modifying data in one table, then three entries would be required. If three users are modifying data in two tables, then six entries would be required. The default value assumes an average of four tables referenced per transaction. For some systems, this value may not be enough. If you set the value of DML_LOCKS to 0, enqueues are disabled and performance is slightly increased. However, you cannot use DROP TABLE, CREATE INDEX, or explicit lock statements such as LOCK TABLE IN EXCLUSIVE MODE. Oracle holds more locks during parallel DML than during serial execution. Therefore, if your database supports a lot of parallel DML, you may need to increase the value of this parameter.
INSTANCE_NUMBER	If specified, this parameter must have unique values on all instances. In Oracle Parallel Server environments, multiple instances can be associated with a single database service. Clients can override connection load balancing by specifying a particular instance by which to connect to the database. INSTANCE_NAME specifies the unique name of this instance. In a single-instance database system, the instance name is usually the same as the database name.

Table 1–2 Initialization Parameter Notes for Multiple Instances

Parameter	Description and Comments
LOG_ARCHIVE_FORMAT	This parameter is applicable only if you are using the redo log in ARCHIVELOG mode. Use a text string and variables to specify the default filename format when archiving redo log files. The string generated from this format is appended to the string specified in the LOG_ARCHIVE_DEST parameter. You must include the thread number. The following variables can be used in the format: ■ %s: log sequence number ■ %S: log sequence number, zero filled ■ %t: thread number ■ %T: thread number, zero filled Using uppercase letters for the variables (for example, %S) causes the value to be fixed length and padded to the left with zeros. An example of specifying the archive redo log filename format is: ■ LOG_ARCHIVE_FORMAT = "LOG%s_%t.ARC"
MAX_COMMIT_ PROPAGATION_ DELAY	This is an Oracle Parallel Server-specific parameter. However, you should not change it except under a limited set of circumstances specific to the Oracle Parallel Server. This parameter specifies the maximum amount of time allowed before the system change number (SCN) held in the SGA of an instance is refreshed by the log writer process (LGWR). It determines whether the local SCN should be refreshed from the lock value when getting the snapshot SCN for a query. Units are in hundredths of seconds. Under unusual circumstances involving rapid updates and queries of the same data from different instances, the SCN might not be refreshed in a timely manner. Setting the parameter to zero causes the SCN to be refreshed immediately after a commit. The default value (700 hundredths of a second, or seven seconds) is an upper bound that allows the preferred existing high performance mechanism to remain in place. If you want commits to be seen immediately on remote instances, you may need to change the value of this parameter.
NLS_* parameters	There are several NLS parameters as described in *Oracle8i Reference* . You can set different values for different instances.
PARALLEL_SERVER	To enable a database to be started in Oracle Parallel Server mode, set this parameter to TRUE in the instance initialization file (`init_`*`sid`*`.ora`).

Table 1–2 Initialization Parameter Notes for Multiple Instances

Parameter	Description and Comments
PARALLEL_SERVER_ INSTANCES	Set this parameter equal to the number of instances in your Oracle Parallel Server environment. Oracle uses the value of this parameter to size memory structures to optimize performance. PARALLEL_SERVER_INSTANCES is an Oracle Parallel Server parameter that specifies the number of instances currently configured. You must set this parameter for every instance. Normally you should set this parameter to the number of instances in your Oracle Parallel Server environment. Oracle uses the value of this parameter to compute the default value of the LARGE_POOL_SIZE parameter when the PARALLEL_AUTOMATIC_TUNING parameter is set to TRUE. A proper setting for this parameter can improve memory use.
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. PROCESSES specifies the maximum number of operating system user processes that can simultaneously connect to Oracle. Its value should allow for all background processes such as locks, job queue processes, and parallel execution processes. The default values of the SESSIONS and TRANSACTIONS parameters are derived from this parameter. Therefore, if you change the value of PROCESSES, you should evaluate whether to adjust the values of those derived parameters. If you do not use defaults, you may want to increase the values for some of the above parameters to allow for additional LCKn and other optional background processes.
For the eight parameters just described in this table, if you do not use defaults, you may want to increase the values for some of these parameters. This allows Oracle to create additional LCKn processes and other background processes to improve performance.	
RECOVERY_PARALLELISM	To speed up the roll forward or cache recovery phase, you may want to set this parameter to specify the number of processes to participate in instance or crash recovery. A value of zero or one indicates that recovery is to be performed serially by one process.

Table 1–2 Initialization Parameter Notes for Multiple Instances

Parameter	Description and Comments
ROLLBACK_SEGMENTS	Use this to specify the private rollback segments for each instance by allocating one or more rollback segments by name to an instance. If you set this parameter, the instance acquires all of the rollback segments named in this parameter, even if the number of rollback segments exceeds the minimum number required by the instance, calculated from the ratio of: TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT. You cannot change the value of this parameter dynamically, but you can change its value and then restart the instance. Although this parameter usually specifies private rollback segments, it can also specify public rollback segments if they are not already in use. To find the name, segment ID number, and status of each rollback segment in the database, query the data dictionary view DBA_ROLLBACK_SEGS.
THREAD	If specified, this parameter must have unique values on all instances. THREAD is an Oracle Parallel Server parameter that specifies the number of the redo thread to be used by this instance. When you create a database, Oracle creates and enables thread 1 as a public thread (one that can be used by any instance). You must create and enable subsequent threads using the ADD LOGFILE THREAD clause and ENABLE THREAD clause of the ALTER DATABASE statement. The number of threads you create is limited by the MAXINSTANCES parameter specified in the CREATE DATABASE statement. In exclusive mode, thread 1 is the default thread. However, you can specify THREAD for an instance running in exclusive mode if you want to use the redo log files in a thread other than thread 1. In parallel mode, you can specify any available redo thread number, as long as that thread number is enabled and is not in use by another instance. A value of zero specifies that this instance can use any available, enabled public thread.
SESSIONS_PER_USER	Each instance maintains its own SESSIONS_PER_USER count. If SESSIONS_PER_USER is set to 1 for a user, the user can log on to the database more than once as long as each connection is from a different instance.

The MTS_DISPATCHER Parameter and Oracle Parallel Server

To enable a Multi-Threaded Server configuration, set the MTS_DISPATCHERS parameter in the common file. The MTS_DISPATCHERS parameter may contain many attributes.

Oracle recommends that you configure at least the PROTOCOL and LISTENER attributes. PROTOCOL specifies the network protocol for which the dispatcher generates a listening end point. LISTENER specifies an alias name for the listeners with which the PMON process will register dispatcher information. The alias should be set to a name that is resolved through a naming method such as a `tnsnames.ora` file.

> **See Also:** *Oracle8i Parallel Server Setup and Configuration Guide* and the *Net8 Administrator's Guide* for complete information about configuring the MTS_DISPATCHER parameter and its attributes and for configuring the Multi-Threaded Server.

LM_* Initialization Parameters

Distributed Lock Manager capacity is determined by the values Oracle sets for the LM_RESS and LM_LOCKS parameters. Table 1–3 describes these parameters. The Distributed Lock Manager automatically calculates values for LM_RESS and LM_LOCKS.

If your shared pool runs out of space, or if the maximum utilization shown in the V$RESOURCE_LIMIT view is greater than the values Oracle sets for these parameters, adjust LM_RESS and LM_LOCKS as described in Chapter 10. Otherwise, you do not need to set these parameters. If you adjust the settings, Oracle recommends that you set the values for them identically across all instances to simplify administration.

Table 1–3 LM_* Initialization Parameters

Parameter	Description
LM_RESS	This parameter controls the number of resources that can be locked by the Distributed Lock Manager (DLM). This parameter includes non-PCM resources such as the number of lock resources allocated for DML, DDL, data dictionary cache locks, and file and log management locks.
LM_LOCKS	This parameter controls the number of locks. Where N is the total number of nodes: LM_LOCKS = LM_RESS + (LM_RESS * (N - 1)) / N

Use increased values for LM_RESS and LM_LOCKS if you plan to use parallel DML or DML performed on partitioned objects.

Part II

Oracle Parallel Server Administration

Part Two describes the general administrative tasks that you must consider for an Oracle Parallel Server environment. This part contains the following chapters:

2 Parallel Execution in Oracle Parallel Server Environments

This chapter discusses parallel execution and its use in Oracle Parallel Server environments. This chapter includes the following sections:

Note: You must set parallel execution parameters before instance startup.

Parallel Execution in Oracle Parallel Server

Although parallel execution does not require Oracle Parallel Server, some aspects of parallel execution described in this chapter apply only to Oracle Parallel Server environments.

See Also:

- *Oracle8i Parallel Server Concepts* for more information about parallel execution on Oracle Parallel Server.
- *Oracle8i Data Warehousing Guide* for more information about using parallel execution.

Setting the Degree of Parallelism

When you set the degree of parallelism, consider all of the available CPUs in your cluster. If you use the parallel automatic tuning feature, Oracle sets the other parallel execution-related parameters for you.

See Also: *Oracle8i Data Warehousing Guide* for more information about Parallel Automatic Tuning and setting the degree of parallelism.

Parameters for Parallel Execution on Oracle Parallel Server

Two parameters affect parallel execution in Oracle Parallel Server environments:

- INSTANCE_GROUPS
- PARALLEL_INSTANCE_GROUP

Use these parameters together to control how parallel execution uses resources in Oracle Parallel Server environments.

Allocating Resources with Instance Groups

Instance groups simplify administration and allow you to more effectively control which instances participate in parallel execution. Instance groups are sets of instances that you designate for specific purposes, such as parallel execution, OLTP, or data warehousing operations.

For example, you could create instance groups such that between 9 a.m. and 5 p.m. users can access instance group B, but after 5 p.m. they use group D. You could also have users access group C for normal OLTP inserts and updates but have users access group D for large parallel tasks to avoid interfering with OLTP performance.

You must define all instance groups before starting up your database. You cannot dynamically add or delete instances from groups. Because instance groups do not incur significant overhead, you can define an unlimited number of groups. You are also not required to use the instance groups once you define them. An instance can belong to more than one group and groups can overlap one another.

For parallel execution, if you do not use instance groups, Oracle determines which instances participate in parallel execution. Oracle does this based on disk affinity and the number of running instances. The instance from which you initiate a parallelized SQL statement need not be a member of the instance group processing the statement. However, the parallel coordinator runs on the instance from which you submit the SQL statement.

Specifying Instance Groups

To specify the instance group or groups to which an instance belongs, set the INSTANCE_GROUPS initialization parameter. Do this by listing the names of the instance groups after the parameter and include these entries within the parameter files of each instance you wish to associate to the group or groups. Thus, the INSTANCE_GROUPS parameter simultaneously defines a group and adds the current instance to it.

For example, to define that instance 1 is a member of groups A and B, place the following entry in your database initialization file for instance 1:

```
INSTANCE_GROUPS = GROUPA, GROUPB
```

To define that instance 2 is a member of groups A and C, place the following entry in your database initialization file for instance 2:

```
INSTANCE_GROUPS = GROUPA, GROUPC
```

As a result, instances 1 and 2 both belong to instance group A, but they also belong to other groups.

Note: INSTANCE_GROUPS is a static parameter; you cannot alter it during a session.

Defining Parallel Instance Groups

Use the PARALLEL_INSTANCE_GROUP parameter to define which instance groups participate in parallel operations. As with the INSTANCE_GROUPS parameter, the default for PARALLEL_INSTANCE_GROUP is a group comprised of all running instances.

To use a particular instance group for a given parallel operation, specify the following in the common parameter file:

```
PARALLEL_INSTANCE_GROUP = GROUPNAME
```

where *GROUPNAME* is the name of the instance group you designate for parallel operations.

All parallel operations initiated from the instance with this entry in its parameter file spawn processes only within the group defined by GROUPNAME. Unlike settings for INSTANCE_GROUPS, you can change the value for PARALLEL_INSTANCE_GROUP using an ALTER SESSION or ALTER SYSTEM statement. However, you can only use PARALLEL_INSTANCE_GROUP to refer to one instance group. 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.

Instance Group Example

In this example, instance 1 has the following settings in its parameter file:

```
INSTANCE_GROUPS = GROUPA, GROUPB
PARALLEL_INSTANCE_GROUP  GROUPB
```

Instance 2 has the following settings in its parameter file:

```
INSTANCE_GROUPS = GROUPB, GROUPC
PARALLEL_INSTANCE_GROUP = GROUPC
```

If you enter the following statements on instance 1, Oracle uses the instances in group GROUPB for parallel execution. Because both instances 1 and 2 are members of group GROUPB, Oracle spawns two server processes on instance 1 and two server processes on instance 2.

```
ALTER TABLE TABLE PARALLEL (DEGREE 2);
SELECT COUNT(*) FROM TABLE;
```

However, if you enter the following statements on instance 1, Oracle uses group GROUPC for parallel execution by spawning two server processes on instance 2 only.

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = 'GROUPC';
SELECT COUNT (*) FROM TABLE;
```

This is because instance 1 is not a member of parallel instance group GROUPC.

To make all instances participate in parallel operations, use a blank within single quotes when declaring the parallel instance group. For example, if you enter the following statements on instance 1, Oracle uses the default instance group, or all currently running instances, for parallel processing. Two server processes spawn on instance 1, and two server processes spawn on instance 2.

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = '';
SELECT COUNT(*) FROM TABLE;
```

Listing Members of Instance Groups

To see the members of instance groups, query the GV$PARAMETER view and examine entries for the INSTANCE_GROUPS parameter.

> **See Also:** *Oracle8i Reference* for complete information about initialization parameters and views.

Other Resource Management Features of Parallel Execution

The other features of parallel execution that optimize resource use are:

- Parallel Execution Load Balancing
- Parallel Execution Adaptive Multi-User

Parallel Execution Load Balancing

Parallel execution load balancing spreads server processes across instances to balance loads. This improves load balancing for parallel execution and parallel DML 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 execution.

Affinity nodes have loads that are about 10 to 15 percent higher than non-affinity nodes. The load balancing feature uses your vendor-specific cluster manager software to communicate among instances. On Massively Parallel Processing systems, Oracle first populates affinity nodes before populating non-affinity nodes.

Parallel Execution Adaptive Multi-User

As workloads on Oracle Parallel Server systems change, the adaptive multi-user feature alters the degree of parallelism for in coming SQL statements based on the perceived workload across the entire cluster. Oracle adjusts the degree of parallelism based on the number of instances and the current workloads of each instance. To enable this feature, set the PARALLEL_ADAPTIVE_MULTIUSER parameter to TRUE.

If the degree of parallelism for an incoming SQL statement is small enough and if other instances are too busy to accommodate part of the SQL statement, Oracle attempts to place the workload onto one instance instead of dividing it among several. This is because when several instances cooperate to process a SQL statement, they can incur prohibitive intra-node communication costs because instances participating parallel execution must use resources to communicate with each other.

Oracle recommends using the parallel adaptive multi-user feature when users process simultaneous parallel execution operations. If you enable parallel automatic tuning, Oracle automatically sets PARALLEL_ADAPTIVE_MULTI_USER to TRUE.

Avoiding Disk Contention in Parallel Processing

To avoid disk contention during parallel table scans, stripe the tables across the instances. Do this using either operating system striping on the disks or by creating tablespaces that use files on multiple nodes.

> **See Also:** *Oracle8i Data Warehousing Guide* for more information about parallel execution load balancing and the adaptive multi-user feature.

Dynamic Performance Views

Use the following performance views to examine parallel execution activity within an Oracle Parallel Server environment:

- GV$PX_SESSION
- GV$PX_SESSSTAT
- GV$PX_PROCESS
- GV$PX_PROCESS_SYSSTAT

See Also: *Oracle8i Reference* for information on using these views to evaluate parallel execution.

Disk Affinity and Parallel Execution

Disk affinity is available only in systems using a shared nothing approach to disks, making such disks visible globally through an operating system-dependent software layer. Disk affinity determines which instance performs parallelized DML or query operations. Affinity is especially important for parallel DML in Oracle Parallel Server configurations. Affinity information that is consistent across statements improves buffer cache hit ratios and reduces forced reads / writes.

The granularity of parallelism for most parallel DML operations is by partition. For parallel execution, however, granularity is by rowid. Parallel DML operations need partition-to-instance mapping to implement affinity. The segment header of the partition is used to determine the affinity of the partition for Massively Parallel Processing 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 Oracle Parallel Server 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 Oracle Parallel Server / MPP configurations.

See Also:

- *Oracle8i Concepts* for a description of Parallel Data Manipulation Language (PDML) and degree of parallelism.
- For a discussion of PDML tuning and optimizer hints, see *Oracle8i Designing and Tuning for Performance*.
- Also refer to each installation and configuration guide for operating system-specific information on disk affinity.

3 Oracle Parallel Server Database Creation Issues

This chapter describes issues surrounding the creation of Oracle Parallel Server databases. Information in this chapter supplements information presented in the *Oracle8i Parallel Server Setup and Configuration Guide* for using the Database Creation Assistant. Topics in this chapter include:

- Creating a Database for Multi-Instance Environments
- Database Objects to Support Multiple Instances
- Changing The Values for CREATE DATABASE Options

Note: The Database Configuration Assistant creates the objects you need for an Oracle Parallel Server database. If the database that the Database Configuration Assistant creates does not match your requirements, Oracle recommends creating your database with the Database Configuration Assistant and modifying it instead of manually creating your database.

See Also: *Oracle8i Parallel Server Setup and Configuration Guide* for information on database creation procedures.

Creating a Database for Multi-Instance Environments

This section covers aspects of database creation specific to Oracle Parallel Server such as:

- Setting Initialization Parameters for Database Creation
- Setting CREATE DATABASE Options

Setting Initialization Parameters for Database Creation

As described in Chapter 1, certain initialization parameters that are critical for database creation or that affect certain database operations must have the same value for every instance. Be sure the settings for these parameters are identical across all instances before creating an Oracle Parallel Server database.

Using ARCHIVELOG Mode

To enable the archiving process (ARCH) 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 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 ARCHIVELOG mode.

You cannot use the STARTUP command to change the database archiving mode. Instead, after you create the database, use the following commands to change to archiving mode and reopen the database with Parallel Server enabled:

```
ALTER DATABASE CLOSE;
ALTER DATABASE ARCHIVELOG;
SHUTDOWN;
STARTUP;
```

Setting CREATE DATABASE Options

This section describes the following CREATE DATABASE options specific to Oracle Parallel Server.

- Setting MAXINSTANCES
- Setting MAXLOGFILES and MAXLOGMEMBERS

- Setting MAXLOGHISTORY
- Setting MAXDATAFILES

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.

For Oracle Parallel Server, 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. The MAXLOGMEMBERS option specifies the maximum number of members or copies per group. For Parallel Server, 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 Oracle Parallel Server.

For Oracle Parallel Server, 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, Oracle overwrites the oldest entries. The default for MAXLOGHISTORY is zero, which disables log history.

Setting MAXDATAFILES

The MAXDATAFILES option is generic, but Oracle Parallel Server 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 CREATE DATABASE and ALTER DATABASE SQL statements.
- "Redo Log History in the Control File" on page 13-7 for more information about redo log groups and members.

Database Objects to Support Multiple Instances

To prepare a new database for Oracle Parallel Server, create and configure the additional database objects as described under the following headings:

- Creating Additional Rollback Segments
- Configuring the Online Redo Log for Oracle Parallel Server
- Providing Locks for Added Data Files

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 separate tablespace. Do not store these rollback segments in 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 the instance initialization 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 an error message and prevents the instance from starting up.

Using Public Rollback Segments

Any instance can create public rollback segments that are available for any instance to use. Once a rollback segment is in use by an instance, it is only used by that instance until the instance shuts down. When it shuts down, the instance that used the rollback segment releases it for use by other instances.

To create public rollback segments, use the SQL statement CREATE PUBLIC ROLLBACK SEGMENT. Public rollback segments are owned as PUBLIC in the data dictionary view DBA_ROLLBACK_SEGS. If you do not set a value for the ROLLBACK_SEGMENTS parameter for an instance, the instance uses public rollback segments. The procedures you use to create and manage rollback segments are the same regardless of whether Parallel Server is enabled or disabled.

Typically, the parameter file for a 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.

A public rollback segment comes online when an instance acquires it at startup. However, starting an instance that uses public rollback segments does not ensure that the instance uses a particular public rollback segment. The exception to this is when the instance acquires all available public rollback segments.

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 recreate it, you must prevent other instances that require public rollback segments from starting up.

Monitoring Rollback Segments

To monitor rollback segments, query the dynamic performance views V$ROLLNAME and V$ROLLSTAT for information about the current instance's rollback segments. You can also query the data dictionary views

DBA_ROLLBACK_SEGS and DBA_SEGMENTS or the global dynamic views GV$ROLLNAME and GV$ROLLSTAT for rollback segment information.

To monitor rollback segments on another instance, use the command CONNECT *@instance-path* to change the current instance before using the MONITOR command or querying the V$ views.

To list the rollback segments currently in use by an instance, query DBA_ROLLBACK_SEGS with the following syntax:

```
   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, or "online", as shown in the following sample output:

```
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:

- *Oracle8i Administrator's Guide* for more information about rollback segments, and about connecting to a database.
- *Oracle8i Reference* for a description of DBA_ROLLBACK_SEGS and DBA_SEGMENTS, and for information about other dynamic performance views.

Configuring the Online Redo Log for Oracle Parallel Server

Each database instance has its own "thread" of online redo, consisting of its own online redo log groups. When running Oracle Parallel Server, 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 Oracle Parallel Server.

You must create each thread with at least two redo log files (multiplexing), 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. 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 you set THREAD to 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.

```
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.

See Also: *Oracle8i Parallel Server Concepts* for more information about threads of redo.

Disabling Threads

Disable a public or private thread with the ALTER DATABASE DISABLE THREAD statement. 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. You cannot drop or archive these log files. In this case, disable the thread, even though it is already disabled, then re-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 Oracle Parallel Server 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 Oracle Parallel Server 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 13-2.

Providing Locks for Added Data Files

If you add data files while Oracle Parallel Server is running, evaluate whether enough locks are available to cover the new files. Data files that you add in this way use any unassigned locks that were created when Oracle Parallel Server initially created locks to accommodate the value for GC_FILES_TO_LOCKS.

If the remaining number of locks is inadequate to protect the new files and avoid contention, provide more locks by increasing the value for the GC_FILES_TO_LOCKS parameter. Performance problems are likely if you neglect to make these adjustments. This is especially true if your database experiences a high number of inserts. Note, however, that in read-only databases extra locks are unnecessary even if you added many new data files.

If you determine that you need more locks, do the following:

1. Shut down your database.
2. Modify the GC_FILES_TO_LOCKS initialization parameter to provide enough locks for the additional data files.
3. Restart the system.

See Also: "Tips for Setting GC_FILES_TO_LOCKS" on page 9-6.

Changing The Values for CREATE DATABASE Options

You can use the CREATE CONTROLFILE statement to change the value of the following database parameters for a 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.

4 Administering Instances

This chapter describes starting up and shutting down instances. It discusses the following topics:

- Starting Up and Shutting Down Instances
- How Instances Are Affected by SQL*Plus and SQL

Starting Up and Shutting Down Instances

This chapter discusses these topics in the following sections:

- Starting Instances
- Shutting Down Instances

Starting Instances

This section provides procedures for starting instances with Oracle Parallel Server enabled or disabled by explaining:

- Enabling Oracle Parallel Server and Starting Instances
- Setting and Connecting to Instances

Enabling Oracle Parallel Server and Starting Instances

You can start instances using either SQL*Plus or the Oracle Enterprise Manager as described in the *Oracle8i Parallel Server Setup and Configuration Guide*.

Note: In Oracle8*i*, the keywords SHARED, EXCLUSIVE, and PARALLEL are obsolete in the STARTUP and ALTER DATABASE MOUNT statements.

Starting an Instance Using SQL*Plus

Starting Oracle Parallel Server instances is procedurally identical to starting a single instance with these exceptions:

1. Make sure the PARALLEL_SERVER parameter is set to TRUE in the instance initialization file.
2. Ensure your Cluster Manager software is running. Detailed instructions on Cluster Manager software administration appear in your operating-system specific documentation. If the Cluster Manager is not available or if Oracle cannot communicate with this component, Oracle displays the error ORA-29701: "Unable to connect to Cluster Manager".

   See Also: For Windows NT, the Cluster Manager is described in your vendor documentation.

3. Start any required operating system specific processes. For more information about these processes, see your Oracle operating system-specific documentation.

If you use the Oracle Database Configuration Assistant to create your database, specifically request that it create an Oracle Parallel Server database so it sets PARALLEL_SERVER to TRUE in your initialization file. To start a database with Oracle Parallel Server disabled, use the default value of FALSE for the PARALLEL_SERVER parameter on your instance.

> **See Also:** *Oracle8i Parallel Server Concepts* for more information about Cluster Manager.

Using RETRY 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, use the STARTUP RETRY statement. This causes the new instance to retry mounting the database every five seconds until it succeeds or has reached the retry limit. Use the syntax:

```
STARTUP OPEN database_name RETRY
```

To set the maximum number of times the instance attempts to mount the database, use the SQL*Plus 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 using the RETRY will not repeat connection attempts. 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.

Setting and Connecting to Instances

Before you can set instances and connect to them, you must install and configure Net8 for the Oracle Parallel Server nodes and any clients that access these nodes. This allows clients to establish remote connections to the nodes.

> **See Also:**
>
> - *Oracle8i Parallel Server Setup and Configuration Guide.*
> - *Net8 Administrator's Guide.*

SQL*Plus commands operate on the current instance with some exceptions as noted under the next heading. The current instance can be either the local, default instance on which you initiated your SQL*Plus session, or it can be a remote instance. Because the SQL*Plus prompt does not indicate which instance is the current instance, be sure you direct your commands to the correct instance.

Initiating a SQL*Plus session and connecting to the database without specifying an instance directs all SQL*Plus commands to the local instance. In this case, the default instance is also the current instance.

To switch the current instance from the local instance to a remote instance, do one of the following:

- Re-execute the CONNECT command specifying a remote instance net service name as in this example:

```
CONNECT SYSTEM/MANAGER@net_service_name
```

- Disconnect from the database and execute a SET INSTANCE command as in this example:

```
SET INSTANCE net_service_name
```

Then issue another CONNECT command with only your user ID and password. Specifying a remote instance with the CONNECT command while connected to the database by way of an instance allows you to switch from one instance to another without disconnecting.

See Also:

- *Net8 Administrator's Guide* for information on configuring net service names.
- Your operating system-specific Oracle documentation for more information about the exact format required for the connect string used in the SET INSTANCE and CONNECT commands.

The SET INSTANCE and SHOW INSTANCE Commands

When using SET INSTANCE to specify an instance on a remote node for the STARTUP command, the parameter file for the remote instance must be accessible by the local node.

The SHOW INSTANCE command displays the net service name for the current instance. SHOW INSTANCE returns the value "LOCAL" if you have not used SET INSTANCE during the SQL*Plus session.

To reset to the default instance, use SET INSTANCE without specifying a net service name or specify "LOCAL". Do not follow the SET INSTANCE command with the word "DEFAULT"; this indicates a connect string for an instance named "DEFAULT".

The CONNECT Command

Connecting as SYSOPER or SYSDBA allows you to perform privileged operations, such as instance startup and shutdown. Multiple SQL*Plus sessions can connect to the same instance at the same time. SQL*Plus automatically disconnects you from the first instance whenever you connect to another one.

See Also:

- *Oracle8i Parallel Server Setup and Configuration Guide* for the proper specification of *net_service_name*.
- *Oracle8i Administrator's Guide* for information on connecting to the database using SYSDBA or SYSOPER privileges.

Shutting Down Instances

Shutting down an Oracle Parallel Server instance is procedurally identical to shutting down a single instance with these exceptions:

- In Parallel Server, shutting down one instance does not interfere with the operation of other running instances.
- To shut down a database mounted in shared mode, shut down every instance in the Parallel Server cluster.
- When you shut down an instance 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 displays 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 displays 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 you issue 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 needing it.

- If multiple SQL*Plus sessions are connected to the same instance simultaneously, all but one must disconnect before the instance can shut down normally. You can use the IMMEDIATE or ABORT option of the SHUTDOWN command to shut down an instance when multiple SQL*Plus sessions (or any other sessions) are connected to it.

  See Also: *Oracle8i Administrator's Guide* for more information on shutting down Oracle databases.

How Instances Are Affected by SQL*Plus and SQL

The following topics are described in this section:

- How SQL*Plus Commands Apply to Instances
- How SQL Statements Apply to Instances

How SQL*Plus Commands Apply to Instances

Table 4–1 describes how common SQL*Plus commands apply to instances.

Table 4–1 How SQL*Plus Commands Apply to Instances

SQL*Plus Command	Associated Instance
ARCHIVE LOG	Always applies to the current instance.
CONNECT	Applies the default instance if no instance is specified in the CONNECT command.
HOST	Applies to the node running the SQL*Plus session, regardless of the location of the current and default instances.
RECOVER	Does not apply to any particular instance, but rather to the database.
SHOW INSTANCE	Displays information about the current instance, which can be different from the default local instance if you have redirected your commands to a remote instance.
SHOW PARAMETER and SHOWN SGA	Display parameter and SGA information from the current instance.
STARTUP and SHUTDOWN	Always apply to the current instance. These are privileged SQL*Plus commands.

Note: The security mechanism that Oracle uses when you execute privileged SQL*Plus 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 operating system-specific documentation.

How SQL Statements Apply to Instances

Most 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 also 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, use the ALTER SYSTEM ARCHIVE LOG CURRENT statement. The THREAD option of ALTER SYSTEM ARCHIVE LOG allows you to archive online redo log files for a specific instance.

Part III

Oracle Parallel Server Design and Deployment

Part Three explains the design issues for deploying Oracle Parallel Server applications. It includes the following chapters:

5

Application Analysis and Partitioning

This chapter explains application design optimization techniques for Oracle Parallel Server. It includes the following sections:

- Overview of Development Techniques
- Application Transactions and Table Access Patterns
- Selecting A Partitioning Method
- Application Partitioning Techniques
- Departmental and User Partitioning
- Departmental and User Partitioning
- Physical Table Partitioning
- Transaction Partitioning
- Scaling Up and Partitioning
- Adding Instances
- Design-Related Batch Processing Issues

Note: If you have determined that you cannot partition your application, refer to Chapter 8.

Overview of Development Techniques

Application deployment for Oracle Parallel Server requires that you consider special topics and use particular techniques beyond those you would use to develop applications in single-instance environments. This chapter explains these issues and provides a high-level development methodology for deploying parallel server-based applications to optimize Oracle Parallel Server's features.

A poorly written application that provides sub-optimal performance on a single instance will likely run even worse in an Oracle Parallel Server environment. Before moving an application to Oracle Parallel Server from a single instance, tune it as much as possible. Note that even a well-tuned application may run worse on Parallel Server. This is usually due to excessive pinging.

Note: Use the information in the discussions of how to deploy Oracle Parallel Server applications in this section as well as discussions for single-instance application development as described in the Oracle8*i* Application Developers' book set.

Before You Begin, Determine Your Application's Suitability

Before developing Oracle Parallel Server applications, determine whether your system is suitable for multi-instance environments. In general, if you can easily partition your data to avoid inter-instance contention, the more likely Oracle Parallel Server is a suitable solution for you.

The optimal type of application to deploy on Oracle Parallel Server is one that only uses read-only tables, such as Decision Support Systems (DSS). These applications are "perfectly partitionable", thus, inter-instance contention is minimal.

The majority of OLTP applications, however, can be partitioned to varying degrees. Because of the introduction of Cache Fusion, you may experience significant performance gains despite being unable to partition your data to create strict data-to-instance affinities. However, partitioned applications are more scalable than non-partitioned applications.

Note: Simple, updatable star schemas and non-partitionable applications can experience performance problems when deployed with Oracle Parallel Server.

How Detailed Must Your Analysis Be?

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

A primary characteristic of high performance Oracle Parallel Server systems is that they minimize the computing resources used for Parallel Cache Management. This means they minimize the number of instance lock operations. In addition, the machine-to-machine high speed interconnect traffic should remain within, or preferably well below, the design limitations of the cluster.

Application Transactions and Table Access Patterns

Before beginning your analysis, you must understand how Oracle Parallel Server processes various transactions within tables based on transaction types such as:

- Read-Only Tables
- Random SELECT and UPDATE Tables
- INSERT, UPDATE, or DELETE Tables

Read-Only Tables

With tables that are predominantly read-only, all Oracle Parallel Server nodes quickly initialize the PCM locks to shared mode and very little lock activity occurs. Ideally, each read-only table and its associated index structures should require only one PCM lock. This is why read-only tables offer better performance and scalability with Oracle Parallel Server.

Also consider putting read-only tables into read-only tablespaces by using the SQL statement ALTER TABLESPACE READ ONLY. This has several advantages:

- It speeds up recovery
- PCM locks are not required
- You only need to back up a tablespace once after you make it read-only

Scalability of parallel execution in Oracle Parallel Server is subject to the interconnect speed between the nodes. You may also need to use higher degrees of parallelism just to keep the processors busy. It is not unusual to run a degree of parallelism equal to three times the number of nodes or processors.

See Also: *Oracle8i Data Warehousing Guide* for information about setting the degree of parallelism.

Random SELECT and UPDATE Tables

Random SELECT and UPDATE tables have transactions that may read and then update any rows in your tables. This type of access requires multiple lock conversions. First, the instance executing the transaction must obtain a shared PCM lock on one or more data blocks. This lock request may cause lock downgrade operations 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 the same range of data blocks concurrently and frequently, there can be a noticeable response time performance penalty. In some cases you can reduce contention by controlling the lock granularity or the access frequency.

In large tables, however, hardware and practical limitations may mean that the number of fixed PCM locks you can effectively use is limited. In these cases, releasable locks may be a good alternative.

See Also: "Implementing High or Low Granularity Locking" in the Case Study on page A-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 mentioned in the previous section.

Performance issues for randomly modified tables may arise when indexes need to be updated or maintained. This is especially true when inserts are performed or when Oracle searches for free space and then allocates it to a table or index.

For INSERT, DELETE, and UPDATE transactions that modify indexed keys, you need to maintain the table’s indexes. This process requires access to multiple index blocks such as root, branch, and leaf blocks. Thus, the number of potential lock conversions increases. The branch or leaf blocks of an index may split, requiring multiple locks to be held simultaneously. This increases the probability of conflicts across instances. The dispersion of key values in an index can be very important. With monotonically increasing index keys, a hot spot may be created in the right edge of the index key.

If the INSERT and DELETE operations are subject to long-running transactions, then there is a greater chance that another instance will require read consistency information to complete its transactions. This type of concurrence is handled efficiently by Cache Fusion for reads.

Index block contention can be problematic when using a sequence number generator to generate unique keys for a table from multiple Oracle Parallel Server 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.

Creating Reverse Key Indexes

Creating reverse key indexes can improve performance in an Oracle Parallel Server environment where modifications to the index are concentrated on a small set of leaf blocks. A reverse key index reverses the bytes of each indexed column (except the rowid) while keeping the column order. By reversing the keys of the index, the insertions become distributed across all leaf keys in the index.

For example, to create a reverse key index use the syntax:

```
CREATE INDEX i ON t (a,b) REVERSE;
```

Where "a" is the instance number and "b" is a generated unique key.

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 arranged closely 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 8.
- *Oracle8i Concepts.*

Selecting A Partitioning Method

To implement applications that optimize Oracle Parallel Server, use one of these partitioning methods:

- Application Partitioning Techniques
- Departmental and User Partitioning
- Physical Table Partitioning
- Transaction Partitioning

Partitioning is an important part of Oracle Parallel Server deployment because it is the best way to avoid global data block contention or "pinging". With Cache Fusion, partitioning becomes less critical since the number of forced writes to disk significantly decreases.

Partitioning tables to increase Oracle Parallel Server performance has various development and administration implications. From a development perspective, when you partition a table, the quantity and complexity of application code required for that table may increase. In addition, partitioning a table may compromise the performance of other application functions, such as batch and data warehouse queries.

You must also understand your system's performance implications and be aware of the design trade-offs due to partitioning. Your goal is to minimize the need for synchronization. With minimal lock conversions, and the resulting decrease in Distributed Lock Manager activity, Oracle Parallel Server performance is predictable and scalable.

By partitioning applications and data, you can maintain data-to-node affinities. If your system experiences excessive Distributed Lock Manager lock activity, your partitioning strategy may be inappropriate, or the database creation and tuning process was ineffective.

Regardless of the method you use, the method must be "data-centric" to achieve optimal results, as described in the next section.

Partitioning Based on Data, Not Function

Focus your partitioning strategy on the data and how it is used, not on the application's functions. When you partition, load balancing is not necessarily the primary objective.

To determine which partitioning method to use, examine the data access properties of your business function, for example, the locality, type, and frequency. Group them into a few main categories and determine the locking strategy and configuration.

Using this method to set up partitioning creates data block-to-cache affinities across all instances. In addition, Distributed Lock Manager activity is more predictable so you can more easily achieve:

- Higher availability with minimum lock allocation
- Greater speed-up and scale-up with minimal lock conversions

If the methodologies described in this chapter do not provide adequate performance, consider doing one of the following:

- Creating lock groups
- Using releasable locks
- Using free lists to partition data as described in Chapter 8

Note that the more blocks you cover with a single lock, the greater the likelihood that your application will experience excessive false pings.

Application Partitioning Techniques

One of the simplest ways to partition your database load is to run subcomponents of applications that access the same database on different nodes of the cluster. For example, one subcomponent may only reference a fixed set of tables residing in one set of data files. Another application may reference different tables residing in a different set of data files.

In this example, you can run these applications on different nodes of a cluster and achieve good performance. Moreover:

- There will be few conflicts for the same database objects
- Data block-to-instance affinity will be high
- Global conflicts will be decreased because of the disjoint data set that each subcomponent uses

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 performance, better overall SQL performance, reduced network traffic, and fewer data replication issues.

With this approach, ensure that each application’s tables and indexes are stored such that PCM locks do not cover data blocks used by both applications. Otherwise the benefit of partitioning is lost. To do this, store each application’s table and index data in separate data files.

Applications sharing 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.

Methodology for Application Partitioning

This section describes the following five steps for application partitioning:

- Step 1: Define the Major Functional Areas of the System
- Step 2: Identify Table Access Requirements and Define Overlaps
- Step 3: Define the Access Type for Each Overlap
- Step 4: Identify Transaction Volumes
- Step 5: Classify Overlaps

Step 1: Define the Major Functional Areas of the System

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

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

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

Step 2: Identify Table Access Requirements and Define Overlaps

Determine which tables each functional area accesses and identify the overlaps. Overlaps are simply tables that users from more than one functional area access. Table 5–1 shows the overlapping tables from this example in bold; the remaining tables are accessed exclusively by the listed functions denoted by the column headings.

Table 5–1 Example of Overlapping Tables

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

The objective is to identify overlaps that can cause global conflicts and thus adversely affect application performance. In this example, both functions concurrently access three tables. The remaining tables that are accessed exclusively require fewer locks.

Step 3: Define the Access Type for Each Overlap

Determine the access type for each overlap.

Table 5–2 Example of Table Access Types

Hotel Reservation Operations	Overlap Access Type by Reservations	Overlaps	Overlap Access Type by Front Desk	Front Desk Operations
Table 1	S (Select)	Table 15	S	Table 12
Table 7	I (Insert)	Table 19	I	Table 14
Table 11	U (Update)	Table 20	U	Table 16

In this example, both functions access:

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

Step 4: Identify Transaction Volumes

Estimate the number of transactions you expect the overlaps to generate.

Table 5–3 Example of Table Transaction Volumes

Hotel Reservation Operations	Transaction Overlap by Reservations	Overlaps	Transaction Overlap by Front Desk	Front Desk Operations
Table 1	S **(10 per second)**	Table 15	S **(50 per second)**	Table 12
Table 7	I **(100 per second)**	Table 19	I **(10 per second)**	Table 14
Table 11	U **(50 per second)**	Table 20	U **(90 per second)**	Table 16

Given these transaction volumes, the overlap tables may prove to be a performance problem. However, if the application infrequently accesses the tables, the volumes shown in Table 5–3 may not be a problem.

Step 5: Classify Overlaps

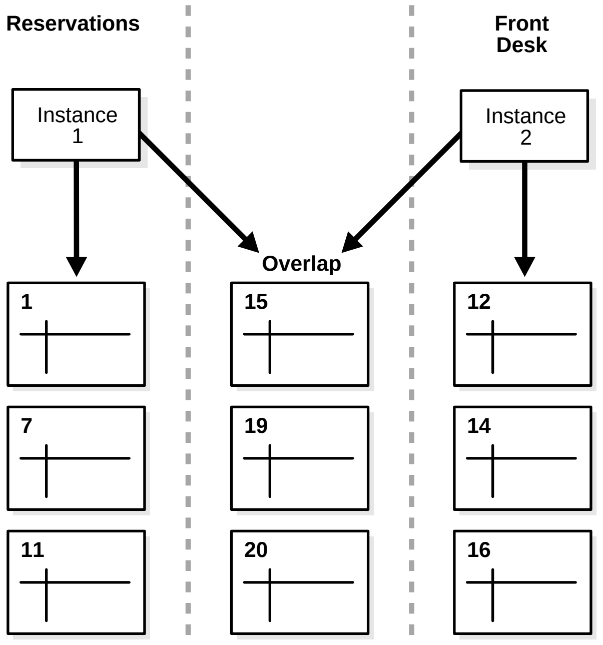
Use the following criteria to determine how to deal with the tables:

- Ignore non-overlapping tables, select-only overlaps, and low-frequency overlaps
- Categorize index-only tables and their indexes
- Categorize mixed access tables and their indexes

Index-only Tables and Their Indexes For these tables, you can assign extents by instance to acquire an exclusive lock. Conflicts for these tables and their indexes are relatively easy for Oracle Parallel Server to resolve.

Resulting Example Configuration Figure 5–1 illustrates the resulting configuration of the tables:

Figure 5–1 Classifying Overlaps for Application Partitioning



If you cannot use application partitioning to resolve conflicts within these tables, consider departmental partitioning as described in the next section.

Departmental and User Partitioning

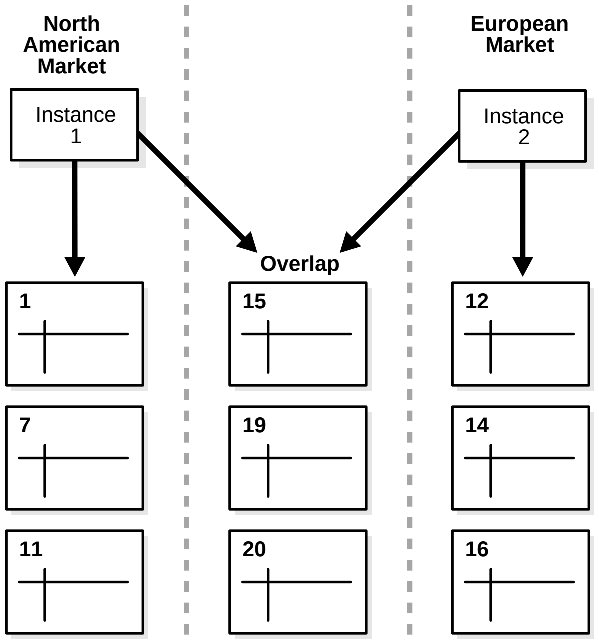
An alternative partitioning method than can help minimize contention is departmental or user partitioning. There are several methods of implementing departmental and user partitioning.

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

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

This configuration might resemble the partitioning illustrated in Figure 5–2:

Figure 5–2 Overlaps for Geographic Partitioning



In addition to geographic partitioning, you can also use the advanced partitioning options of the Oracle8*i* Enterprise Edition. These include three table partitioning methods:

- Range
- Hash
- Composite

Each method has a different set of advantages and disadvantages. Thus, each method is appropriate for a particular situation where the others are not.

Range Partitioning Range partitioning maps data to partitions based on boundaries identified by ranges of column values that you establish for each partition. This feature is generally useful only for Decision Support Systems applications.

Hash Partitioning Hash partitioning maps data to partitions based on a hashing algorithm that Oracle applies to a partitioning key. This feature is primarily useful for DSS applications.

Composite Partitioning Composite partitioning combines the features of range and hash partitioning. With composite partitioning, Oracle first distributes data into partitions according to boundaries established by the beginnings and ends of the partition ranges. Then Oracle further divides the data and distributes it with a hashing algorithm into subpartitions within each range partition.

> **See Also:** *Oracle8i Data Warehousing Guide* for more information about partitioning.

The remaining two partitioning methods discussed in this chapter require significant programming effort and should be used only when the previously described methods do not provide optimal performance.

Physical Table Partitioning

Physical table partitioning involves the division of one table into two or more smaller tables. This requires application changes to use the new names of the smaller tables. Report programs must also change so they can join the smaller tables as needed to provide data.

However, Oracle can automatically manage partition independence for you. That is, if you use the same table name for all the smaller tables across the application, Oracle automatically segregates the data.

If you have adequate resources, you can use transaction partitioning. However, this method is quite complex and it takes much longer to implement than any of the methods described previously.

Transaction Partitioning

Transaction partitioning is the lowest level partitioning method. This method requires a three-tiered architecture where clients and servers are separated by a transaction monitor processing layer. Based on the content of a transaction, the transaction monitor routes transactions that act on specific tables to specific nodes. Using this method, you can create and load your tables using any method because the transaction monitor determines which node processes a particular transaction.

This method also allows you to achieve fine-grained transaction control. This makes transaction processing monitors very scalable. However, significant development effort is required to deploy this method.

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.

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 instance that is running the relevant application. 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 remote procedure call (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.

Scaling Up and Partitioning

If you have properly partitioned your application for Oracle Parallel Server, as the size of your database increases, you should be able to maintain the same partitioning strategy and simultaneously have optimal performance.

The method to use when adding new functionality is dependent upon the types of data the new functions access. If the functions access disjoint data, your existing partitioning scheme should be adequate. If the new functions access the same data as the existing functions, you may need to change your partitioning strategy.

If your application is popular and it attracts more users than you expected, you may need to add more instances. Adding a new instance may also require that you repartition your application. You may also need to repartition if you add new instances in response to your application experiencing increased transaction rates.

Adding Instances

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

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

You must also consider how adding additional instances to accommodate unexpected growth can result in complex, expensive re-partitioning tasks. This is why your initial planning and design process should take long-term growth into account.

Design-Related Batch Processing Issues

When scheduling batch operations, do not assume you can separate batch jobs from online processing and run batch jobs on a separate instance. Instead, consider the batch jobs as part of the normal daily load.

When developing a partitioning strategy, include you application's batch processing needs. Attempt to run batch jobs when there are not a lot of interactive users, such as a night or during off-peak hours.

Note: Beware of batch jobs that perform full scans on shared tables.

Using the DBMS_JOB Package to Manage Batch Job and Instance Affinity

Use this package to control which instances process which jobs. This package allows you to distribute job processing 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 Oracle Parallel Server 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 Oracle Parallel Server instance.

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

See Also:

- *Oracle8i Supplied PL/SQL Packages Reference f*or details about DBMS_JOB.
- *Oracle8i Administrator's Guide.*

6 Database Design Techniques

This chapter describes database design techniques for Oracle Parallel Server environments. The sections in this chapter include:

Principles of Database Design for Oracle Parallel Server

When designing database layouts for shared Oracle Parallel Server databases, remember that accessing globally shared data from multiple nodes increases transaction processing costs. In other words, multi-node transactions incur more wait time and higher CPU consumption than transactions processed on single-node systems. Because of this, carefully consider the database access characteristics of your applications you can create scalable database designs. In general, you can achieve scalable systems by:

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

Many of the principles of the above to points have been covered in Chapter 5. The most scalable and efficient application designs for clustered systems enable a high degree of transaction affinity to the data the transactions access on the nodes. The more that your application' s data access is local and thus does not require cross-instance synchronization, the more efficient your application.

All systems have a certain proportion of data with lower node affinity. This data is shared across the cluster and thus requires synchronization. Cache Fusion reduces the costs associated with globally shared database partitions by more efficiently keeping data synchronized across multiple nodes. This increases the data's availability to all nodes in the cluster. Some features exist that help optimize concurrent access to globally shared data.

Some database resources can become critical when certain transactions execute in an Oracle Parallel Server environment. A high rate of inter-instance access to blocks in the same table can cause increased I/O, messaging, context switches and general processing overhead. If a table has one or more indexes to maintain, the cost may increase even more due to the relative complexity of an index access. Searching for free space and allocating it when inserting new data requires access to space management structures, such as segment free lists. Also, generating unique sequence numbers using an Oracle sequence number can become a severe bottleneck if every node in the cluster uses it.

All resources can be either locally cached on disk or they must be re-generated frequently. Each type of database operation, such as INSERTs, UPDATEs, DELETEs, and SELECTs, require different types of resources depending on whether your application runs in shared or exclusive mode.

Database Operations, Block Types, and Access Control

Most business transactions involve a mixture of INSERTs, UPDATEs, DELETEs, and SELECTs. Exactly what percentage of each of these a transaction uses depends on the business transaction type.

Likewise, each of these operations accesses certain types of data blocks. These block types can be categorized as:

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

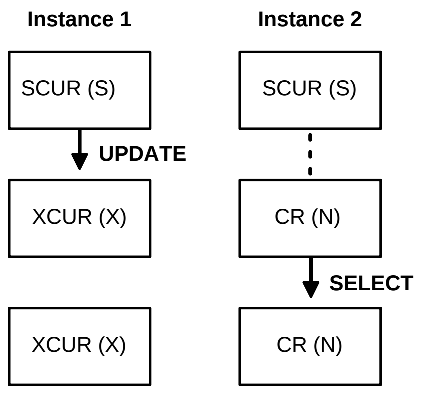
Concurrent access to data blocks in a cache is controlled by access or lock modes. In the buffer cache, a block can be accessed in exclusive current read (XCUR), shared current read (SCUR), or consistent read (CR) mode. To guarantee global cache coherency, these access modes map to global lock modes as shown in Table 6–1:

Table 6–1 Parallel Cache Management Lock Mode and Buffer State

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

Each operation accesses certain types of blocks in a particular mode, as shown in Figure 6–1:

Figure 6–1 Modes of Block Access



Block Accesses During INSERTS

When Oracle processes an INSERT, it reads the segment header of a database object. This might mean that the INSERT must read the segment header of a table or an index to locate a block with sufficient space in which to fit a new row into the segment free list. Therefore, to process INSERTs, Oracle reads the CURRENT, or most up-to-date version of the header block. If there is enough free space in the block after completing the INSERT, the block remains on the free list and the transaction reads the corresponding data block and writes to it. For this sequence of events:

- The segment header is acquired in SCUR mode, which means that the instance must request a global S lock
- The data block is acquired in XCUR mode, or globally in X mode

If the free space in the block is insufficient after inserting, Oracle unlinks the block from the free list. This means Oracle updates the segment header block containing the free list. For this sequence of events:

1. The segment header block is first acquired in SCUR mode (global S lock)
2. After checking the block, Oracle then escalates the buffer access mode to XCUR, (global X)

3. Oracle removes the block from the free list
4. If a new block beyond the current highwater mark is used, Oracle raises the highwater mark
5. The data block is read in XCUR mode and written to disk

This scenario assumes that the highwater mark of the segment, or the last or highest block in the segment containing data, resides in an allocated extent. If no free list groups were defined, the highwater mark as well as a map of allocated extents is stored in the segment header. If Oracle must allocate an additional extent to insert the object, Oracle raises the highwater mark and updates the extent map. In other words, Oracle changes the segment header block in a consistent fashion; this also requires Oracle to lock the header block in exclusive mode.

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

For an insert into a table with an index, even more data block accesses are required. First, Oracle reads the header block of the index segment in SCUR mode, then Oracle reads the root and branch blocks in SCUR mode. Finally, Oracle reads the leaf block in XCUR mode. Depending on the height of the index tree, Oracle would also have to read more branch blocks. If a free list modification is required, Oracle must escalate the index segment header lock mode to XCUR mode. If there is concurrency for the segment header due to free list modifications, the header block can ping back and forth between multiple instances.

Using FREELIST GROUPS at table creation effectively achieves free list partitioning. The number of FREELIST GROUPS that you define for an object should a least match the number of nodes in the cluster that participate in INSERTs. With free list groups, the free lists are in a separate block that is physically located after the header block. Processes inserting rows into the same objects from different nodes must hash to different free list group blocks. Hence, the free list group blocks remain local to a particular instance. This means Oracle does not need to acquire globally conflicting locks.

Static and Dynamic Extent Allocation

There are two methods with which Oracle allocates extents to provide sufficient free space for newly inserted rows. One method requires manual intervention; the other is derived automatically from certain settings. Static allocation requires, for example, that you issue statements such as:

```
CREATE TABLE
STORAGE (FREELIST GROUPS 2)
ALTER TABLE
ALLOCATE EXTENT
FILE SIZE INSTANCE;
```

when creating or altering a table or index. This type of statement allows the allocation of extents of data blocks from particular physical files to free list groups and thus to instances. Note that you must set the parameter INSTANCE_NUMBER to ensure than an instance consistently uses a particular allocation of free list groups. If you do not set a value for INSTANCE_NUMBER, Oracle allocates the space to the object; Oracle does not take the space from a particular free list group. Instead, Oracle uses the master free list in the general segment header.

When Oracle pre-allocates extents in this manner, Oracle contiguously allocates blocks to a particular free list group. Files containing these objects are good candidates for 1:N locks with a blocking factor; in other words, use the ! syntax when you set the GC_FILES_TO_LOCKS parameter and when you set the blocking factor to the extent size allocated. For example, if your extent size is 10 blocks for file number 4 consisting of 1000 blocks, then use the following syntax:

```
GC_FILES_TO_LOCKS3D "43D1000!10"
```

This ensures that the blocks in that extent are covered by one lock.

For dynamic allocations, simply set GC_FILES_TO_LOCKS with a blocking factor, as described in the previous syntax. The space management layer determines the new extent size based on the value for *!n*.

You can use several methods to define this, for example:

```
GC_FILES_TO_LOCKS3D"43D1000!10"
```

or

```
GC_FILES_TO_LOCKS3D"43D1000!10R"
```

or

```
GC_FILES_TO_LOCKS3D"43D0!10"
```

where the first example assigns 10 contiguous blocks to a fixed lock out of 1000 locks pooled for this file. The second example assigns a releasable lock in the same manner, and the third uses releasable locks out of an unlimited pool.

Depending on which method you use, a certain number of contiguous blocks comprise an extent of free space and these blocks are covered by the same lock. The method you use depends on your application's requirements. If ease of use is a high priority and the data files are extensible, use dynamic allocation as shown by the third entry in the previous example set. However, static allocation has the advantage of reducing run-time space management and the required data dictionary table and row cache updates. This is because the extents are already allocated.

In summary, when designing for INSERT-intensive transactions in Oracle Parallel Server, if you have identified certain tables as "insert only" when determining the partitioning strategy, then:

1. Run inserting transactions only from one node
2. Use free list groups on these tables and on any indexes associated with them
3. Use the ! syntax for the GC_FILES_TO_LOCKS parameter when dynamically allocating space
4. Pre-allocate extents to the table or index using the CREATE TABLE... ALLOCATE EXTENTS or ALTER TABLE... ALLOCATE EXTENTS statements and set the blocking factor ! for the GC_FILES_TO_LOCKS parameter to the size of the extents

Block Accesses During UPDATES

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

1. Reads the buffer in XCUR mode and get a global X lock
2. Writes to the buffer and modify a row
3. If the updated row fits into the same block, the instance does not need to acquire new blocks from the free list and the modification is complete; segment header access is unnecessary
4. The instance retains the global X lock until another instance requests a lock on the block in a conflicting mode, so Oracle writes the dirty buffer to disk for the other instance to "see" the most current changes

5. Oracle closes the lock or retains it in NULL mode, and Oracle reads the buffer from disk if the local instance requests the block for subsequent updates; this is known as a "forced read"

If Oracle has built an index on the table, the UPDATE:

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

If the index key value was changed, Oracle:

1. Rereads the root and branch blocks in SCUR mode
2. Reads the leaf block in XCUR mode
3. Modifies the index key value for the updated row

During the update operation with an index, a block can be "pinged" out of the cache of the updating instance at any time and would have to be reacquired. The shared global locks on the root and branch blocks are not an issue, as long as another instance reads only these blocks. If a branch block has to be modified because a leaf block splits, Oracle escalates the S lock to an X lock, thus increasing the probability of conflict.

It is therefore essential to clearly identify frequently updated tables when you establish your partitioning strategy. The update frequency, randomness of access, and data referencing patterns eventually determine the layout and locking strategy you use. For random access, a locking policy using 1:1 releasable locks is appropriate. If certain transactions access the same range of blocks repetitively, you can use a table partitioning and transaction routing strategy. Once data partitions within a table can be established, very few fixed locks are needed.

In the case of frequent random access, the probability of contention for the same data or index blocks can be decreased by setting PCTFREE to a higher value when creating and loading the table, so that a block is populated by fewer rows. However, the trade-off might be more block reads and more I/O to access the same number of rows.

See Also : *Oracle8i Parallel Server Concepts* for more information about 1:1 and 1:n locks, and releasable and fixed lock durations.

Block Accesses During DELETES

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

Consequently, the transaction acquires the segment header or free list group block in exclusive mode. The block in question is returned to the instance’s free list group, if there is one. In general, you should schedule massive deletions to occur during off-peak hours and you should run them from one instance.

Block Accesses During SELECTS

A SELECT reads a buffer in either SCUR or CR mode. For an SCR, such as for segment headers or when the only readers in the system are for a particular block, a global S lock is acquired. When a block is modified or contains uncommitted changes, and a consistent read version is requested, the buffer containing that version will be in CR mode. If the most recent modifications are made on another node, a CR copy from the modifying node must be requested. Once the request has completed, no lock is retained on the instance that received the consistent read version of the block.

For a full table scan, a SELECT may have to read the segment header in order to determine the extent boundaries for the extents allocated to a table. As this requires Shared Current access to the buffer containing the header block, in Parallel Server the global S lock might conflict with an INSERT that attempts to modify the header block.

Global Cache Coherence Work and Block Classes

In general, data blocks, index blocks, rollback segment headers, free list group blocks, rollback segment blocks, and segment headers are considered to be different classes of blocks. All of these classes are subject to "pinging" because they represent structures of a shared database.

For rollback segment headers and rollback segment blocks, the effect of global cache synchronization is less significant, because instances should privately own rollback segments in Oracle Parallel Server. This limits writes to a rollback segment to the local instance. With Cache Fusion for consistent reads, the effect of rollback segment cache coherence operations is limited. Most consistent read information is generated by the local instance which creates a version of a data block and sends it directly to the requesting instance.

Without free list groups and with suboptimal physical storage parameters for an index or a table, segment header blocks can be "pinged" frequently. Configuring free list groups can alleviate this. As a general rule, segment header pings should amount to more than 5% of the total pings.

The V$CLASS_PING view lists the number of lock converts of a particular type for the different block classes. Use this view to monitor the proportion of pings for each block class of the total pings.

General Recommendations for Database Object Parameters

The following represent some general recommendations for database objects:

- For tables or indexes that are expected to grow or shrink during regular OLTP processing, or objects that can have frequent INSERTs or DELETEs from multiple nodes, use FREELISTS and FREELIST GROUPS to build the objects.
- Preallocate extents to the object's FREELIST GROUP and / or use a blocking factor when defining GC_FILES_TO_LOCKS. This ensures that blocks belonging to a particular extent are used by only one instance and that the blocks are covered by the same lock.
- For objects and files with random local and remote access, restrict the number of rows contained in one block by selecting the correct database block size or by allowing more free space when loading data by setting PCTFREE to a higher value (the default is 10%). However, this results in increased space requirements.

Index Issues

In most high volume OLTP systems, inter-instance contention for index blocks increases the cost of Oracle Parallel Server processing and the commonly used B*Tree index structures are vulnerable to "pinging" at various levels. A "right-growing" tree can incur frequent pinging of one particular leaf block. While traversing the tree structure, branch blocks might have to be "forced read", because they were last modified by another instance. Leaf block splits are vulnerable because three blocks need to be modified in one transaction. Generally, some situations you should avoid are:

- Leaf and branch block contention

- Root block contention
- Index segment header contention

The following section addresses how to avoid leaf and branch block contention.

Minimizing Leaf/Branch Block Contention

You can use various strategies to isolate or distribute access to different parts of the index and improve performance.

Use reverse-key indexes to avoid the right-growing index trees. By reversing the keys, you can achieve a broader spread of index keys over the leaf blocks of an index and thus lower the probability of accessing the same leaf and branch blocks from multiple instances. However, reverse key indexes do not allow index range scans, so carefully consider their use.

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

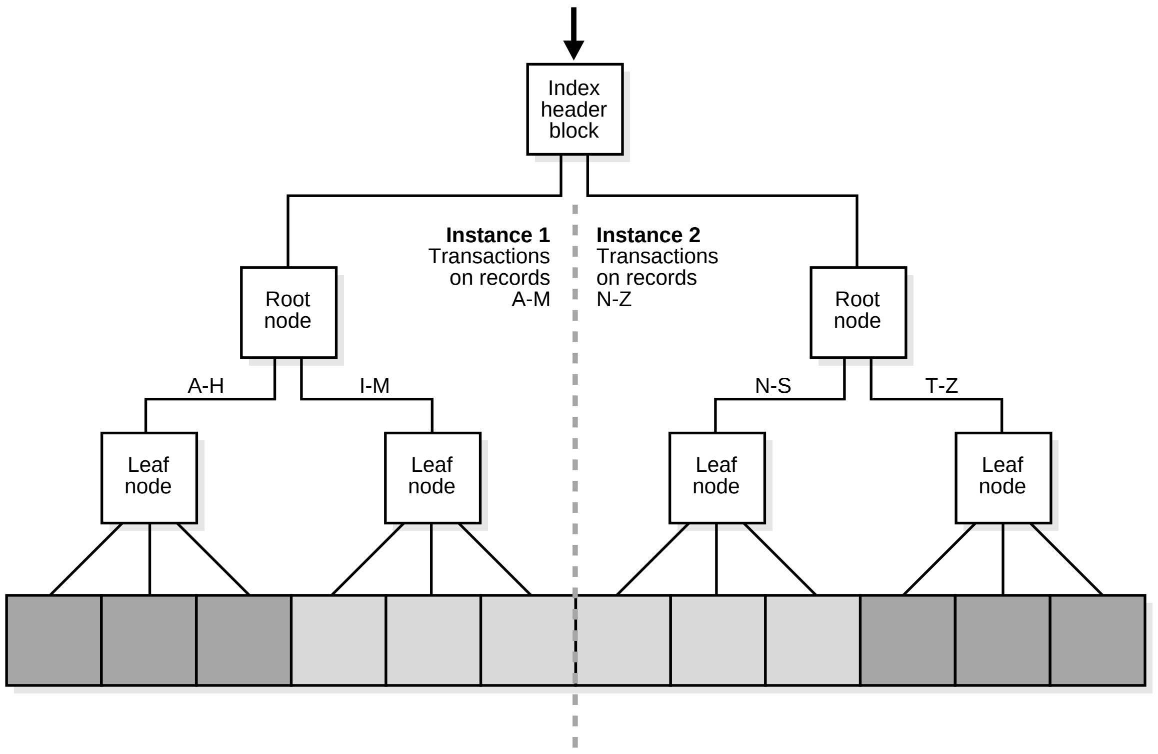
For other sequentially assigned values, adjust the index value and use INSTANCE_NUMBER to generate the index key, as shown in the following example:

$$\textit{index key 3D (instance\_number -1)} * 100000 + \textit{Sequence number}$$

Note: Use local partitioned indexes wherever possible.

Figure 6–2 shows how transactions operating on records stored in tables partitioned by range can minimize leaf and branch block contention.

Figure 6–2 Node Affinity for Transactions Against Tables Partitioned by Range



Locking Policy For Indexes

Determining the optimal locking policy for files that have indexes can be problematic. Usually, the best practice is to use releasable locks. Using fixed locks on indexes increases the potential for false pings. The following guidelines should help you when deciding how to assign locks:

- Avoid a right-growing index tree by using reverse key indexes, if possible, or use techniques to spread the access over multiple leaf blocks
- Partition index access by using either an appropriate index key value that is based on a logical partitioning key or by using local partitioned indexes wherever possible; route transactions to dedicated nodes based on key data values
- If you cannot partition at all, use releasable locks
- If you can use local index partitioning, assign fewer 1:N fixed locks to the files containing the partitions.

See Also: *Oracle8i Parallel Server Concepts* for more information about 1:1 and 1:N locks, and releasable and fixed lock durations.

Using Sequence Numbers

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

```
ALTER SEQUENCE SEQUENCE_NAME CACHE 200;
```

Calculating Sequence Number Cache Size

Base your estimates for sequence number cache size on three factors:

- Transaction rate
- Number of nodes in the cluster
- Stability of the cluster

Using ordering suppresses caching in Oracle Parallel Server. It is normal to lose some numbers after executing the SHUTDOWN command. It is also not unusual to experience a complete loss of sequence numbers after instance failures.

External Sequence Generators

You should implement external sequence generators when:

- Ordering is essential
- Accountability for all numbers is required
- There are high transaction rates
- There are many instances
- There are many sequence numbers

Note: Avoid using database tables to hold sequence numbers.

Detecting Global Conflicts On Sequences

If sequences are insufficiently cached or not cached at all, severe performance problems may result with an increase in service times. This may also result in reduced scalability.

If you experience performance problems, examine statistics in the V$SYSTEM_EVENT view as described below to determine whether the problem is due to the use of Oracle sequences:

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

Logical And Physical Database Layout

Base a physical partitioning strategy on the following considerations:

- Transaction profiles for objects
  - How often are they accessed?
  - How are they accessed?
- Function dependency for objects, if any
  - Business function
  - Data

You should group these objects into files and tablespaces so that you can apply consistent locking policies.

General Suggestions for Physical Layouts

Before grouping objects into tablespaces, create a pool of raw volumes that you can later assign to tablespaces. Because Oracle Parallel Server must use shared raw volumes or shared raw partitions to physically store data, the available disks should be partitioned into volumes of various sizes. Begin by considering the following recommendations:

1. Create a large pool of shared volumes to be assigned to logical partitions later.
2. Define standard sizes for raw volumes, such as 10M, 100M, 500M, and 1G.
3. Slice or stripe the volumes over as many disks as your initial I/O volume calculation indicates. Take into consideration that 10 to 15% read and write I/O should be added to the calculations for global cache synchronization I/O.
4. Review the plan. You may need to restructure the layout because some files will experience higher I/O volumes than others due to pings.

Your goal should be to create a flexible physical layout. Also make sure you account for the I/O required to preserve cache coherence and ensure that this will not adversely affect I/O latencies.

Tablespace Design

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

- Frequently and randomly accessed tables and indexes with a lower probability of having affinity to certain transactions or portions of the database
- Tables and indexes that are mostly read or read-only and infrequently modified
- Tables and indexes whose data can be viewed as subdivided into largely disjoint and autonomous data sets

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

- Tables should be separated from indexes
- Assign true, read-only tables to READ-ONLY tablespaces.
- Each node should have TEMPORARY tablespaces
- Group smaller reference tables in the same tablespace
- Separate insert-only tables from other tables

Once you have logically grouped your database objects into partitions and physically assigned them to tablespaces, you can develop an efficient and adaptable locking policy as described in the following chapters.

Global Cache Lock Allocation

After completing your tablespace design, there will be a few distinct groupings based on their expected access distribution, estimated access frequency, and affinity to transactions and nodes. These groupings might include:

- Frequently and randomly accessed files with little or no affinity to transactions or nodes and a high probability of false pinging
- Read-only or "read-mostly" files
- Frequently accessed files that are shared by multiple nodes but whose data are subdivided into largely autonomous data sets
- Files containing rollback segments and temporary segments

Based on these subdivisions, assign locks as follows:

- Assign releasable locks to files that are frequently and randomly modified by multiple nodes; this locking policy is often the best choice for index files
- Do not assign locks to read-only tablespaces
- Only assign fixed locks to read-mostly data that has a low probability of experiencing access conflicts and that has data portions that are accessed mostly in a disjoint fashion
- Do not assign locks to rollback segments and files that belong to temporary tablespaces. Instead, allocate rollback segment locks by setting GC_ROLLBACK_LOCKS so that only a few fixed locks are necessary

Wherever you assign locks, assigned them as releasable if GC_RELEASABLE_LOCKS provides sufficient resources so that the cost of opening and closing locks can be minimized.

Conclusions And Guidelines

Your response time and throughput requirements ultimately determine how stringent and well-isolated your partitioning strategy needs to be and how much effort you should invest in achieving an optimal design. Cache Fusion removes a significant amount of overhead associated with consistent read transactions that request data other nodes. This allows you to implement a simpler database design and still achieve optimal performance.

A careful analysis of data access distributions in terms of transaction profiles and functional dependencies is the basis for allocating work to particular instances. Moreover, this makes a system more robust and more scalable.

Broadly speaking, 80% or more of overhead results from 20% or less of a given workload. Dealing with the 20% by observing some simple guidelines and caveats can produce real benefits with minimal effort:

- Index block contention should be avoided by all means. However, if index key values are not modified by multiple instances, the overhead may be acceptable.
- Globally shared data access may be acceptable if the probability of remote access and thus false pings is low.
- Long running transaction that modify globally shared data must be avoided and should be run at times of lower system use.
- Read-only tablespaces should be used wherever data remain constant.

- Free list groups should be defined for partitioned as well as nonpartitioned data that are modified frequently.

In general, you should design your database for application partitioning and create the tablespaces to permit data striping. This simplifies the processing for parallel data loads and inter-instance parallel queries.

7 Planning the Use of PCM and Non-PCM Instance Locks

This chapter explains the initialization parameters you set to allocate Parallel Cache Management (PCM) locks and non-PCM locks to data files in Oracle Parallel Server environments. It contains the following sections:

- Planning the Use and Maintenance of PCM Locks
- How Oracle Assigns Locks to Blocks
- Examples of Mapping Blocks to PCM Locks
- Non-PCM Instance Locks

Note: Tuning PCM locks may not provide optimal application scalability. For more information, please refer to Chapter 5 and Appendix A.

See Also:

- *Oracle8i Parallel Server Concepts* for a conceptual discussion of PCM locks and GC_* parameters.
- *Oracle8i Reference* for descriptions of initialization parameters used to allocate locks for Oracle Parallel Server.

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
- The Key to Allocating PCM Locks
- Examining Data Files and Data Blocks

Planning and Maintaining Instance Locks

The Distributed Lock Manager (DLM) 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 the locks and resources require. In planning locks, consider the following issues:

- If you attempt to use more locks than the number configured in the DLM facility, Oracle displays an error message and shuts down the instance.
- Changing the GC_* or LM_* initialization parameters to specify large numbers of locks affects the amount of memory used and the amount of memory available in the System Global Area.
- The number of instances also affects the memory requirements and number of locks needed by your system.

The Key to Allocating PCM Locks

The key to allocating PCM locks is to analyze how often data is changed using the INSERT, UPDATE, and DELETE statements. 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 entire tablespace is read-only, you can assign it a single lock. If you did not assign locks to the tablespace, the system attempts 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.

The 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 5 and Chapter 6.

Examining Data Files and Data Blocks

You must allocate locks at various levels by specifying:

- The maximum number of PCM locks to allocate for all data files.
- How many locks to allocate to blocks in each data file.
- Particular locks to cover particular classes of datablocks in a given file.

Begin by examining your data files and the blocks they contain.

Determining File ID, Tablespace Name, and Number of Blocks

Use the following statement 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;
```

Oracle displays results 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
. . .
```

Determining the Number of Locks 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 data files, but a query-only application does not need many locks; a single lock on the data file 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).

How Oracle Assigns Locks to Blocks

This section explains how fixed locks and releasable locks are assigned to blocks. (1:1 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 System Global Area 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, count the number of actual fixed locks that 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 responds with something similar to the following if you 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 file 1 and file 5 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, file 1 and file 2 have the same value for START_LCK indicating that they share the locks. File 5 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. 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* x 2), where *n* represents the rollback segment number. A value of *n* = 0 indicates the 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 x 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 releasable 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 fixed to lock element numbers as follows:

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

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

(DBA) modulo $(locks\_in\_class)$

Examples of Mapping Blocks to PCM Locks

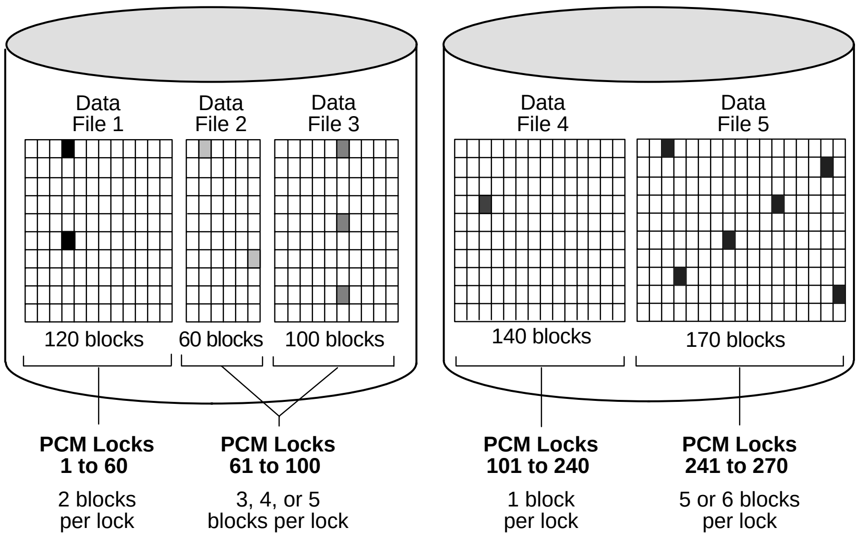
- Setting GC_FILES_ TO_LOCKS
- Sample Settings for Fixed Locks with GC_FILES_TO_LOCKS
- Sample Releasable Setting of GC_FILES_TO_LOCKS

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 data files.

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

Figure 7–1 Mapping PCM Locks to Data Blocks



Example 1 Figure 7–1 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 data file 1 shown in Figure 7–1, 60 PCM locks map to 120 blocks, which is a multiple of 60. Each PCM lock therefore covers two data blocks.

In data files 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 data file 2, and two or three data blocks in data file 3. Thus, one PCM lock may cover three, four, or five data blocks across both data files.

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

In data file 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 7–1 can be held in either read-lock mode or read-exclusive mode.

Example 2 The following parameter setting allocates 500 PCM locks to data file 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 data files.

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

Example 4 The following parameter value allocates 200 PCM locks each to files 1 through 3; 50 PCM locks to data file 4; 100 PCM locks collectively to data files 5, 6, 7, and 9; and 20 data locks in contiguous 50-block groups to data files 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 data files 5, 6, 7, and 9 covers one or more data blocks in each data file, unless a data file contains fewer than 100 data blocks. If data files 5 to 7 contain 500 data blocks each and data file 9 contains 100 data blocks, then each PCM lock covers 16 data blocks: one in data file 9 and five each in the other data files. Alternatively, if data file 9 contained 50 data blocks, half of the PCM locks would cover 16 data blocks (one in data file 9); the other half of the PCM locks would only cover 15 data blocks (none in data file 9).

The 20 PCM locks assigned collectively to data files 8 and 10 cover contiguous groups of 50 data blocks. If the data files 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 data file 8 or data file 10, but not in both. This is because each of these PCM locks covers 50 contiguous data blocks. If the size of data file 8 is not a multiple of 50 data blocks, then one PCM lock must cover data blocks in both files. If the sizes of data files 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.

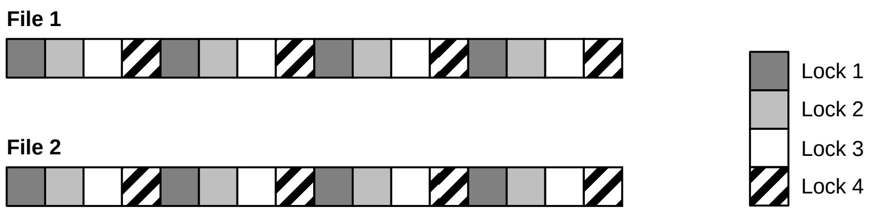
Sample Settings for Fixed Locks with GC_FILES_TO_LOCKS

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.

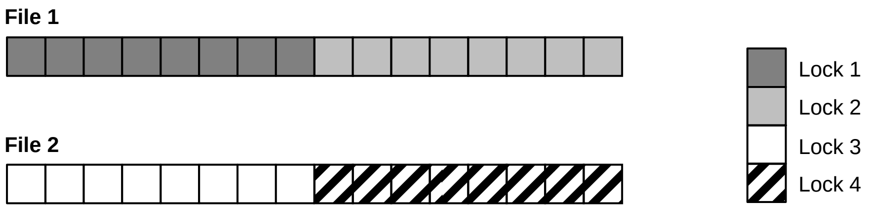
Figure 7–2 GC_FILES_TO_LOCKS Example 5



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

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

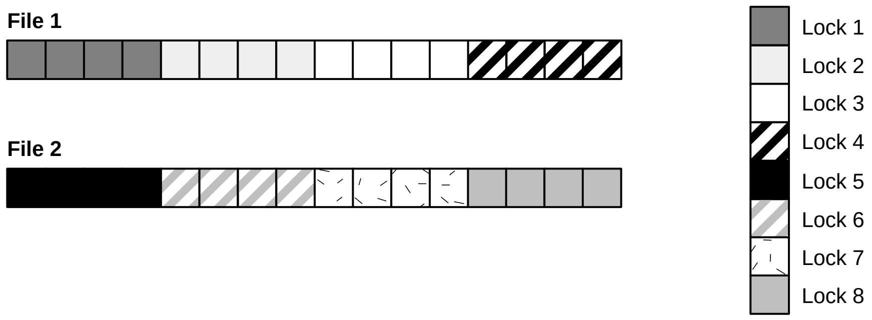
Figure 7–3 GC_FILES_TO_LOCKS Example 6



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

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

Figure 7–4 GC_FILES_TO_LOCKS Example 7



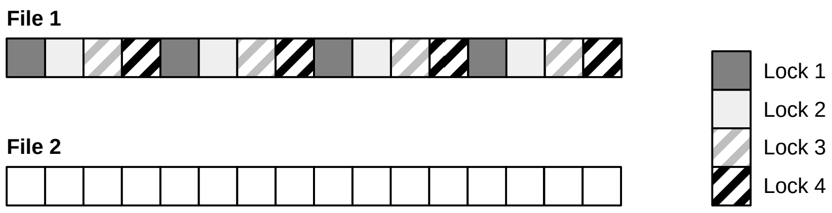
Sample Releasable Setting of GC_FILES_TO_LOCKS

The following example shows releasable locking mixed with fixed locking.

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

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

Figure 7–5 GC_FILES_TO_LOCKS Example 8



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 a key issue. Every time Oracle converts a lock from one mode to the other, you need locks. Consider the interaction of different instances on a table. Also consider the number of rows in a block, the number of rows in a table, and the table’s growth rate. Based on this analysis, group your objects into files, and assign free list groups.

Figure 7–6 PCM Lock Worksheet 1

Object	**Operations needing X mode: Writes**			**Oracle Parallel Server needing S mode: Reads**	**TS/Data File**
	INSERTS	**UPDATES**	**DELETES**	**SELECTS**	
A		80%		20% Full table scan? Single row?	
B				100%	
C					
D					

Figure 7–7 PCM Lock Worksheet 2

Object	**Instance 1**	**Instance 2**	**Instance 3**

D	INSERT
	UPDATE
	DELETE
		SELECT	

E			
F			

Figure 7–8 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 an Oracle 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 DLM 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 data files 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 data files 1, 2, and 3, while Instance Y primarily updates data in data files 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 data files 3 and 4, I/O increases. This is because both instances regularly use the same set of PCM locks.

Non-PCM Instance Locks

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

- Overview of Non-PCM Instance Locks
- Transaction Locks (TX)
- Table Locks (TM)
- System Change Number (SCN)
- Library Cache Locks (L[A-Z]), (N[A-Z])
- Dictionary Cache Locks (Q[A-Z])
- Database Mount Lock (DM)

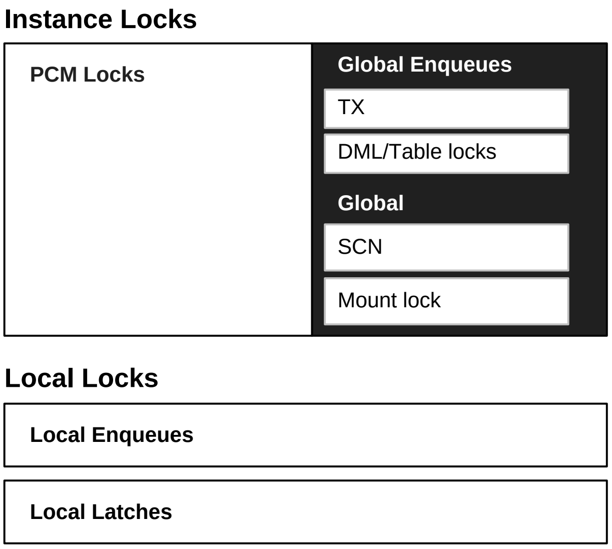
See Also: Chapter 10 for details on how to calculate the number of non-PCM resources and locks to configure in the DLM.

Overview of Non-PCM Instance Locks

This section 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 7–9 highlights non-PCM locks in relation to other locks used in Oracle.

Figure 7–9 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.

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 performs a COMMIT or ROLLBACK. SMON also acquires it in exclusive mode when recovering (undoing) a transaction. Transaction locks are used as a queuing mechanism for processes awaiting the release of an object locked by a transaction in progress.

Table Locks (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, Oracle displays error ORA-61 and the mount attempt fails. Table locks are acquired during the execution of a transaction when referencing a table with a DML statement so that the object is not dropped or altered during the execution of the transaction. This occurs if and only if the DML_LOCKS parameter is 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 that have disabled table locks.

> **See Also:** "Minimizing Table Locks to Optimize Performance" on page 10-3 for more information about minimizing instance locks and disabling table locks for improved performance.

System Change Number (SCN)

The System Change Number (SCN) is a logical timestamp that 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. Oracle increments this value in response to many database events, most notably after COMMITs. A process incrementing the global SCN obtains the SCN lock in exclusive mode, increments the SCN, writes the lock value block, and downgrades the lock. Access to the SCN lock value is batched. Oracle keeps a cache copy of the global SCN in memory. A process may obtain 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 operating system-specific documentation for information about SCN implementation.

Library Cache Locks (L[A-Z]), (N[A-Z])

When a database object (table, view, procedure, function, package, package body, trigger, index, cluster, synonym) is referenced during parsing or during the 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 you drop a table, 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, 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 an instance has mounted a particular database. This lock is only used with Oracle Parallel Server. It is the only multi-instance lock used by Oracle Parallel Server in exclusive mode and prevents another instance from mounting the database in shared mode.

In Oracle Parallel Server single shared mode, a DM lock is held in shared mode, so another instance can mount the same database in shared mode. In Oracle Parallel Server exclusive mode, however, another instance cannot to obtain the lock.

8 Using Free List Groups to Partition Data

This chapter explains how to allocate free lists and free list groups to partition data. Free lists exist in single-instance Oracle. However, free list groups only exist in Oracle Parallel Server environments. You use free list groups in Oracle Parallel Server to partition data to minimize contention for free space.

Only use free list groups to partition data when your application profile does not allow table partitioning. It is much simpler to use partitioned tables and indexes to accomplish the same thing that free list groups accomplish.

> **See Also:** Chapter 5 explains how to use partitioned tables and indexes.

Topics in this chapter include:

> **See Also:** *Oracle8i Parallel Server Concepts* for a conceptual overview of free list groups.

Overview of Free List Implementation Procedures

Use the following procedures to use free lists 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 clauses in the CREATE statements 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 can improve the performance of an application that initially appears not to be ideally suited to Oracle Parallel Server.

Deciding How to Partition Free Space for Database Objects

Use the worksheet in this section to analyze database objects and to 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 the objects into categories as described in this section.

Objects Read-Only Tables

If a table does not have high insert activity or sufficient updates to require new space allocations, the table does not need free lists or free list groups.

Objects in Partitioned Applications

With proper partitioning of certain applications, only one node needs to insert into a table or segment. In such cases, free lists may be necessary if there are many users. Free list groups are not necessary if there are few users.

Objects Relating to Partitioned Data

Multiple free lists and free list groups are not necessary for objects with partitioned data.

Objects in Tables 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.

Free Space Worksheet

List each of your database objects, such as tables, clusters, and indexes, in a worksheet as shown in Table 8–1, and plan free lists and free list groups for each.

Table 8–1 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.

Using the CREATE Statement FREELISTS and FREELIST GROUPS Parameters

This section covers the following topics:

- FREELISTS Parameter
- FREELIST GROUPS Parameter
- Creating Free Lists for Clustered Tables
- Creating Free Lists for Indexes

Create free lists and free list groups by specifying the FREELISTS and FREELIST GROUPS storage parameters in CREATE TABLE, CLUSTER or INDEX statements. Do this while accessing the database in either exclusive or shared mode.

A general rule is to create a free list group for an Oracle Parallel Server instance if the instance experiences significant amounts of DML. Then set a value for FREELIST GROUPS equal to the number of instances in the cluster.

Note: Once you have set these storage parameters you cannot change their values with the ALTER TABLE, CLUSTER, or INDEX statements.

FREELISTS Parameter

FREELISTS specifies the number of free lists in each free list group. The default and minimum value of FREELISTS is 1. The maximum value depends on the data block size. If you specify a value that is too large, 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 Parameter

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 equal to the number of instances in Oracle Parallel Server. 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: 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 Clustered Tables

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

Clusters are an optional method of storing data in groups of tables having common key columns. Related rows of two or more tables in a cluster are physically stored together within the database to improve access time. Oracle Parallel Server allows clusters (other than hash clusters) to use multiple free lists and free list groups.

Some hash clusters can also use multiple free lists and free list groups if you created them with a user-defined key for the hashing function and the key is partitioned by instance.

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

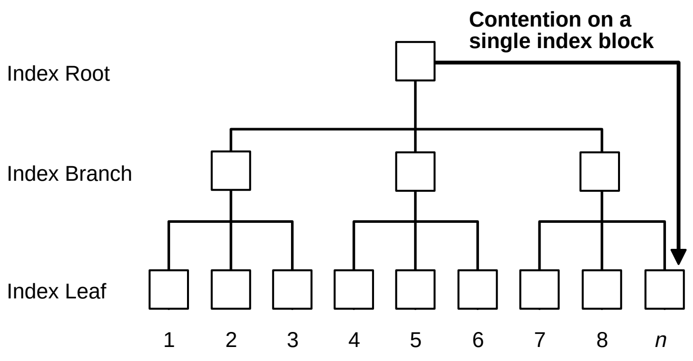
See Also: *Oracle8i SQL Reference* for more information on the REUSE STORAGE clause of the TRUNCATE TABLE statement.

Creating Free Lists for Indexes

You can use the FREELISTS and FREELIST GROUPS storage parameters of the CREATE INDEX statement to create multiple free space lists for concurrent user processes. Use these parameters in the same manner as described for tables.

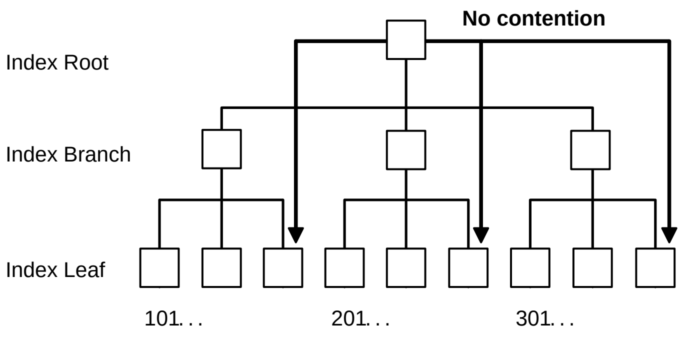
When multiple instances concurrently insert rows into a table having an index, contention for index blocks decreases performance unless index values can be separated by instance. Figure 8–1 illustrates a situation where all instances are trying to insert into the same index leaf block (*n*).

Figure 8–1 Contention for One Index Block



To avoid this, have each instance insert into its own tree, as illustrated in Figure 8–2.

Figure 8–2 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 to 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 clauses with the INSTANCE_NUMBER initialization parameter to associate extents of data blocks with instances.
SET INSTANCE clause	You can use the SET INSTANCE clause of the ALTER SESSION statement to ensure a session uses the free list group associated with a particular instance regardless of the instance to which the session is connected. For example: ALTER SESSION SET INSTANCE = *inst_no*

The SET INSTANCE clause is useful when an instance fails and users 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 data file, use the GC_FILES_TO_LOCKS parameter to allocate specific ranges of PCM locks to each extent, so each PCM lock set is associated with only one group of free lists.

> **See Also:** *Oracle8i Parallel Server Concepts* for more information about associating free lists with instances, users, and locks.

Pre-Allocating Extents

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 Clause
- Setting MAXEXTENTS, MINEXTENTS, and INITIAL Parameters
- Setting the INSTANCE_NUMBER Parameter
- Examples of Extent Pre-Allocation

The ALLOCATE EXTENT Clause

The ALLOCATE EXTENT clause of the ALTER TABLE or ALTER CLUSTER statement enables you to pre-allocate an extent to a table, index or cluster with parameters to specify the extent size, data file, and a group of free lists.

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 parameter This parameter of the ALLOCATE EXTENT clause is the extent size in bytes, rounded up to a multiple of the block size. If you do not specify SIZE, 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 parameter This parameter specifies the data file from which to take space for the extent. If you omit this parameter, space is allocated from any accessible data file 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 parameter This parameter 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 parameter, 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: "Shutting Down Instances" on page 4-5 for more information about the INSTANCE parameter.

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 10K for a block size of 2048 bytes).

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

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

To associate a data block with a free list group, either bring it below PCTUSED by a process running on an instance using that free list group, or specifically allocate it 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 an instance and ensure that it uses the extents allocated to it for inserts and updates. This ensures that it does not use space allocated for other instances. The instance cannot use data blocks in another free list unless another 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 data file 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 parameter, multiple instances share the new space allocation. In this example, every third instance to start up is associated with the same group of free lists.

Example 3 The following CREATE TABLE statement creates a table named 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 the ALTER TABLE statement 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* parameter 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 Data Block Address to Lock Name
- !blocks with ALLOCATE EXTENT Syntax

Translation of Data Block 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 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 available each time an instance requires the allocation of blocks.

See Also: Chapter 9 for more information about the ALLOCATE EXTENT syntax.

Identifying and Deallocating Unused Space

This section covers:

- Identifying Unused Space
- Deallocating Unused Space
- Space Freed by Deletions or Updates

Identifying Unused Space

The DBMS_SPACE package contains procedures with 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 begin allocating space again. Create the package using the DBMSUTIL.SQL script as described in the *Oracle8i Utilities*.

Deallocating Unused Space

Unused space you have allocated to an instance using the ALLOCATE EXTENT command cannot be deallocated. This is 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 the ALTER TABLE or ALTER INDEX statement to trim the segment to the high water mark.

Space Freed by Deletions or Updates

Blocks freed by deletions or by updates that decreased the size of rows go to the free list and free list group of the process that deletes them.

9 Setting Instance Locks

This chapter explains how to set instance locks. It contains the following topics:

Setting GC_FILES_TO_LOCKS: PCM Locks for Each Data File

Set the GC_FILES_TO_LOCKS initialization parameter to specify the number of Parallel Cache Management (PCM) locks covering data blocks in a data file or set of data files. This section covers:

- GC_FILES_TO_LOCKS Syntax
- Fixed Lock Examples
- Releasable Lock Examples
- Guidelines for Setting GC_FILES_TO_LOCKS

Note: Whenever you add or resize a data file, create a tablespace, or drop a tablespace and its data files, adjust the value of GC_FILES_TO_LOCKS before restarting Oracle with Parallel Server enabled.

See Also: *Oracle8i Parallel Server Concepts* to understand how Oracle determines the number of data blocks that are covered by a single PCM lock.

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 releasable locks will be used instead of fixed 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 locks are releasable: they may be released by the instance when no longer needed. Releasable PCM locks are taken from the pool GC_RELEASABLE_LOCKS.

Note: GC_ROLLBACK_LOCKS uses the same syntax. Do not use spaces within the quotation marks of the GC_FILES_TO_LOCKS parameter.

In addition to controlling the mapping of PCM locks to data files, GC_FILES_TO_LOCKS controls the number of locks in the default bucket. Oracle uses the default bucket for all files not explicitly mentioned in GC_FILES_TO_LOCKS. You can use a value of zero in setting this parameter, 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, also use fixed locks.

You can specify releasable PCM locks by using the R option with the GC_FILES_TO_LOCKS parameter. Oracle takes 1:N releasable PCM locks from the pool of GC_RELEASABLE_LOCKS.

REACH is a keyword that combines "R" with the word "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 data file contains fewer data blocks than the number of PCM locks, some PCM locks will not cover a data block in that data file.

The default value for *!blocks* is 1. When you specify *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 data files. You can use *blocks* to allocate a set of PCM locks to cover a single data file where PCM lock contention on that data file is minimal, thus reducing PCM lock management.

Always set the *!blocks* value to avoid interfering with the 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"
```

Releasable Lock Examples

To specify releasable locks with low granularity 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 locks with a group factor of 4 for file 1.

The following entry indicates that 1000 releasable locks protect file 1 in groups of 25 blocks:

```
GC_FILES_TO_LOCKS = "1=1000!25R"
```

Guidelines for Setting GC_FILES_TO_LOCKS

Use the following guidelines to set the GC_FILES_TO_LOCKS parameter:

- Always specify all data files 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 data file should never exceed the number of blocks in the data file. This ensures that if a data file increases in size, the new blocks can be covered by the extra PCM locks.

  If a data file is defined with the AUTOEXTEND clause or if you issue the ALTER DATABASE... DATAFILE... RESIZE statement, then you should regularly monitor the data file for an increase in size. If the data file's size is increasing, then update the parameter GC_FILES_TO_LOCKS as soon as possible. Then shut down and restart Oracle Parallel Server.

  Note: Restarting Oracle Parallel Server is not required, but if you do not shut down and restart it, 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 data files, the DLM uses some PCM locks to cover more datablocks than specified. This can diminish performance, so always ensure that sufficient PCM locks are available:
- When you add new data files, 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 data file using the ALTER TABLESPACE... ADD DATAFILE statement, with Oracle Parallel Server running. If you do this, update the setting for GC_FILES_TO_LOCKS as soon as possible, then shut down and restart Oracle Parallel Server.
- To reduce resource contention by creating disjoint data to be accessed by different instances, place the data files on different disks. Use

GC_FILES_TO_LOCKS to allocate PCM locks to cover the data blocks in the separate data files.

- Specify relatively fewer PCM locks for blocks containing infrequently modified index data. Place indexes in their own tablespace or in their own data files within a tablespace so a separate set of PCM locks can be assigned to them. Use only one lock for read-only indexes.
- Files not mentioned in GC_FILES_TO_LOCKS use releasable locks.

Tips for Setting GC_FILES_TO_LOCKS

Setting GC_FILES_TO_LOCKS is an important tuning task in Oracle Parallel Server. 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 Data Files 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 data file, create a tablespace, or drop a tablespace and its data files, adjust the value of GC_FILES_TO_LOCKS before restarting Oracle with Parallel Server 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 the V$DATAFILE view 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:

- 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 only one lock to that file. This only works if there is absolutely no writing done to the file or even if changes are made to the blocks, such as those done during block clean out.

Setting Tablespaces to Read-Only

If a tablespace is 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 on the tablespace. The exception to this is a single lock you can assign to ensure the tablespace does not have to contend for spare locks.

Checking File Validity

Determine the number of objects in each file using the following syntax:

```
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 Data Files Without Changing Parameter Values

Consider the consequences for PCM lock distribution if you add a data file 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, Oracle gives the data file locks from the pool of remaining locks, and the file must contend with all files you omit from your setting for 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 that you use 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. Too low a value for GC_RELEASABLE_LOCKS could adversely affect performance.

The statistic "global cache freelist waits" in the V$SYSSTAT view shows the number of times the system runs out of releasable locks. If this occurs, as indicated by a non-zero value for global cache freelist waits, increase the value of GC_RELEASABLE_LOCKS.

Setting GC_ROLLBACK_ LOCKS

If you are using fixed locks, 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(R.BLOCKS) 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. The default setting for GC_ROLLBACK_LOCKS is:

```
GC_ROLLBACK_LOCKS = "0-128=32!8REACH"
```

This protects rollback segments 0 through 129 with locks. The first 129 rollback segments have 32 releasable locks, with a grouping of 8. In other words, each lock covers 8 contiguous blocks.

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 of the following examples is invalid and 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
- Determining How Much Time PCM Lock Conversions Require
- Identifying Sessions That Are Waiting for PCM Lock Conversions to Complete
- This section covers the following topics:

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 assess 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 forced writes';
```

Table 9–1 shows how to interpret the ping rate.

Table 9–1 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 occur. False pinging may be occurring.
> 1	False pings are definitely occurring.

Use this formula to calculate the percentage of false pings:

$$\frac{(ping\_rate - 1)}{ping\_rate} * 100$$

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 forced 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 forced writes';
```

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.

Determining How Much Time PCM Lock Conversions Require

Be sure to check the amount of time needed for a PCM lock acquisition. This time differs across systems. Enter the following SQL statement to find the lock acquisition duration:

```
   SELECT *
   FROM V$SYSTEM_EVENT
   WHERE EVENT LIKE 'global cache%'
```

Oracle responds with output similar to:

```
EVENT                    TOTAL_WAITS TOTAL_TIMEOUTS TIME_WAITED AVERAGE_WAIT
---------------------------------------------------------------------------
global cache lock open s         743              0         494    .66487214
global cache lock open x        5760              0        5945   1.03211806
global cache lock null to s      263              0         697   2.65019011
global cache lock null to x     2149              0        7804   3.63145649
global cache lock s to x        1427              0        1394   .976874562
global cache cr request        25248              5        4729   .187301965
global cache lock busy            21              0          46   2.19047619
global cache bg acks               2              0           0            0
```

Identifying Sessions That Are Waiting for PCM Lock Conversions to Complete

Enter the following SQL statement to determine which sessions are currently waiting and which have just waited for a PCM lock conversion to complete:

```
   SELECT *
   FROM V$SESSION_WAIT
   WHERE EVENT LIKE 'global cache%' AND 'wait_time = 0'
```

PCM and Non-PCM Lock Names and Formats

This section covers the following topics:

- Lock Names and Lock Name Formats
- PCM Lock Names
- Non-PCM Lock Names

Lock Names and Lock Name Formats

Oracle names all enqueues and instance locks using one of the following formats:

- *type ID1 ID2*
- *type*, *ID1*, *ID2*
- *type* (*ID1*, *ID2*)

where:

type	A two-character type name for the lock type, as described in the V$LOCK table.
ID1	The first lock identifier, used by the DLM. The convention for this identifier differs from one lock type to another.
ID2	The second lock identifier, used by the DLM. The convention for this identifier differs from one lock type to another.

For example, a space management lock might be named ST00. 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. Buffer Cache Management Locks are of type "BL". The syntax of PCM lock names is *type ID1 ID2*, where:

type	Is always BL because PCM locks are buffer locks.
ID1	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.
ID2	The block class.

Some example PCM lock names are:

BL (100, 1)	This is a data block with lock element 100.
BL (1000, 4)	This is a segment header block with lock element 1000.
BL (27, 1)	This is an undo segment header with rollback segment #10. The formula for the rollback segment is 7 + (10 * 2).

Non-PCM Lock Names

Non-PCM locks have many different names. Table 9–2 contains a list of the names:

Table 9–2 Non-PCM Lock Types and Names

Type	Lock Name
CF	Controlfile Transaction
CI	Cross-Instance Call Invocation
DF	data file
DL	Direct Loader Index Creation
DM	Database Mount
DX	Distributed Recovery
FS	File Set
KK	Redo Log "Kick"
IN	Instance Number
IR	Instance Recovery

Table 9–2 Non-PCM Lock Types and Names

Type	Lock Name
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
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: *Oracle8i Reference* for descriptions of non-PCM locks.

10

Ensuring DLM Capacity for Locks and Resources

To reduce contention for shared resources and to gain maximum Oracle Parallel Server performance, ensure that the Distributed Lock Manager is adequately configured for all the locks and resources that your system requires. This chapter covers the following topics:

- Overview of Planning Distributed Lock Manager Capacity
- Planning Distributed Lock Manager Capacity
- Adjusting Oracle Initialization Parameters
- Minimizing Table Locks to Optimize Performance

When you have finished adjusting your system's settings for locks and resources, you can use SQL*Loader to load data into your database as described under the topic:

- Using SQL*Loader

See Also: *Oracle8i Parallel Server Concepts* for an overview of Parallel Cache Management and non-Parallel Cache Management locks.

Overview of Planning Distributed Lock Manager Capacity

Planning the allocation of Parallel Cache Management (PCM) locks alone is not sufficient to manage locks on your system. Besides explicitly allocating PCM locks, you must continually monitor the Distributed Lock Manager to ensure it is adequately configured. You must do this 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 locks 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 Distributed Lock Manager Capacity

Distributed Lock Manager capacity is determined by the settings for the LM_RESS and LM_LOCKS parameters. The Distributed Lock Manager automatically calculates values for LM_RESS and LM_LOCKS based on other parameter settings in your initialization parameter file. The settings that Oracle makes for LM_RESS and LM_LOCKS appear in your `alert.log` file at startup. These settings, however, are only estimates because enqueue resource usage is application dependent.

If your shared pool runs out space, or if the maximum utilization shown in the V$RESOURCE_LIMIT view is greater than the values Oracle sets for LM_RESS and LM_LOCKS, increase your settings for LM_RESS and LM_LOCKS and re-examine the statistics in V$RESOURCE_LIMIT. Otherwise, you do not need to set LM_RESS and LM_LOCKS.

Avoiding Dynamic Allocation of Resources and Locks

If the required number of locks or resources increases beyond the amount Oracle initially allocates, Oracle will allocate additional locks and resources from the System Global Area 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 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 accurately 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.

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

Do not, however, specify actual parameter values considerably greater than needed for each instance. Setting these parameters unnecessarily high incurs overhead.

Minimizing Table Locks to Optimize Performance

Obtaining table locks, such as DML locks, for inserts, deletes, and updates can diminish performance in Oracle Parallel Server. Locking a table in Oracle Parallel Server is undesirable because all instances holding locks on the table must release those locks. Consider disabling these locks entirely using one of the two methods described under the following headings:

- Disabling Table Locks
- Setting DML_LOCKS to Zero

Note: If you disable all table locks, you cannot perform DDL commands against either the instance or the table.

Disabling Table Locks

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

```
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 statement:

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

Once you execute this syntax, all currently executing transactions commit before enabling the table lock. The statement does not need to wait for new transactions starting after issuing the ENABLE statement.

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.

Setting DML_LOCKS to Zero

Table locks are set with the initialization parameter DML_LOCKS. If the DROP TABLE, CREATE INDEX, and LOCK TABLE statements are not needed, set DML_LOCKS to zero to minimize lock conversions and achieve maximum performance.

Note: If DML_LOCKS is set to zero on one instance, you must set it to zero on *all* instances. With other values, this parameter need *not* be identical on all instances.

Using SQL*Loader

Once you have configured your Distributed Lock Manager, you are ready to load data into the database. An efficient method of loading data is to use SQL*Loader.

With SQL*Loader, you can use either conventional or direct path loading for Oracle Parallel Server databases. However, each method has advantages and disadvantages.

The conventional path method imposes identical data integrity rules as Oracle imposes on regular user-based inserts. Redo logs, rollback segments, indexes, and triggers function as they would during normal insert processing. Because overall data integrity is more important than processing speed, Oracle recommends using the conventional path method for data loads unless it is critical that you reduce the duration of load processing.

When using the conventional path method and running loads in parallel, you should avoid:

- Read contention on input files
- Write contention on affected data files
- Saturating the CPU of any node
- Running out of space for the required extents

SQL*Loader's direct path method bypasses most processing within the System Global Area. When SQL*Loader adds rows to tables, Oracle does not record the rows in rollback segments. Oracle therefore cannot create read-consistent blocks for queries from the new data. However, users can read new records after the data is written to disk.

Parallel direct loads can write block images into the same data file block addresses. To avoid this, use the PARALLEL keyword to set a flag in the control file. Each parallel SQL*Loader session checks the flag to ensure there is not a non-parallel direct load running against the same table. This forces Oracle to create new extents for each session.

See Also: *Oracle8i Utilities* for more information on SQL*Loader.

Part IV

Oracle Parallel Server Performance Monitoring and Tuning

Part Four describes how to monitor performance statistics and adjust parameters to improve Oracle Parallel Server performance. It contains the following chapters:

11

General Tuning Recommendations

This chapter provides an overview of Oracle Parallel Server tuning issues by presenting a general methodology with which you can tune your Oracle Parallel Server applications. This chapter covers the following topics:

See Also: *Oracle8i Parallel Server Setup and Configuration Guide* for more information about performance monitoring and tuning with Oracle Parallel Server Management.

Overview of Tuning Oracle Parallel Server

With experience, you can anticipate many performance problems before deploying your Oracle Parallel Server applications. Some single-instance tuning practices are valid when tuning Oracle Parallel Server applications. However, you must also effectively tune the buffer cache, shared pool, and your shared disk subsystems with Parallel Server-specific goals in mind.

Oracle Parallel Server introduces parameters that are not used in single-instance environments. Many of these are tunable parameters that can significantly enhance Parallel Server performance when you properly set them. However, even the most effective tuning cannot overcome problems caused by poor analysis or database and application design flaws.

Tuning Oracle Parallel Server also requires monitoring several views and collecting Parallel Server-specific statistics. Do this using the methods described in this chapter.

Statistics for Monitoring Oracle Parallel Server Performance

This section describes the statistics that you can use for specifically monitoring and tuning Oracle Parallel Server applications. Topics in this section include:

- Contents of V$SYSSTAT and V$SYSTEM_EVENT
- Recording Statistics for Tuning

Oracle maintains many statistics in local System Global Areas. Access these statistics using SQL against dynamic performance views or V$ tables as described in this chapter.

You can also use utilities such as UTLBSTAT and UTLESTAT to record statistics over a period of time to capture them for specific measurement intervals. Statistics are available as message request counters or as timed statistics. Message counters include statistics showing the number of a certain type of lock conversion. Oracle Parallel Server timed statistics reveal, for example, the total or average time waited for read and write I/O on particular operations.

The most significant statistics from an Oracle Parallel Server perspective are:

- Cache-related statistics such as consistent gets, db block gets, db block changes, and waits for busy buffers
- I/O statistics such as physical reads, physical writes, cross-instance writes, and wait times for reads and writes
- Global cache lock requests and wait times, such as global cache gets, global cache converts, and waits for events such as null-to-X conversions

Two of the most important views showing Oracle Parallel Server-specific statistics are V$SYSSTAT and V$SYSTEM_EVENT.

Contents of V$SYSSTAT and V$SYSTEM_EVENT

The following section describes the contents of V$SYSSTAT and V$SYSTEM_EVENT and lists other Oracle Parallel Server-specific views.

Statistics in V$SYSSTAT

The V$SYSSTAT view includes the following statistics:

- consistent gets
- db block gets
- db block changes
- physical reads
- physical writes
- DBWR cross-instance writes
- global cache get
- global cache converts
- global cache get time
- global cache convert time
- global cache cr blocks received

- global cache cr block receive time
- global cache cr timeouts
- global cache convert timeouts
- CPU used by this session

Statistics in V$SYSTEM_EVENT

The V$SYSTEM_EVENT view contains the following statistics:

- db file sequential read
- db file scattered read
- db file parallel write
- log file sync
- global cache lock null to x
- global cache lock null to s
- global cache lock s to x
- global cache lock open x
- global cache lock open s
- global cache cr request
- global cache lock busy
- buffer busy
- buffer busy due to global cache
- enqueue
- row cache
- library cache pin
- lkmgr wait for remote messages

Other Parallel Server-Specific Views

Other important statistics relate to the following performance issues:

- Buffer cache usage
- Types of lock conversions
- Lock activity with regard to block classes and files
- Messages sent and received by the Distributed Lock Manager

These statistics appear in the following views:

- V$CACHE
- V$LOCK_ACTIVITY
- V$CLASS_PING
- V$FILE_PING
- V$DLM_MISC

In addition to these views, you should analyze operating system statistics showing CPU usage, disk I/O, and the amount of CPU used for certain background processes such as LCK, LMD, BSP, DBW using procedures discussed in this chapter.

Recording Statistics for Tuning

Oracle recommends that you record statistics about the rates at which certain events occur. You should also record information about specific transactions within your Oracle Parallel Server environment. To do this, you can use performance-oriented utilities, such as UTLBSTAT and UTLESTAT, that compute statistic counts per second and per transaction. You can also measure the number of statement executions and the number of business transactions that applications submit.

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

This reveals an application profile in terms of the resources used per transaction and the application workload. The counts per transaction are useful for detecting changing workload patterns, while rates indicate the workload intensity.

Performance and Efficiency of Oracle Parallel Server Workloads

In Oracle Parallel Server, application performance and scalability are determined by the rate and cost of synchronization between nodes. You can measure the costs by identifying how effectively a transaction uses CPU resources. A transaction uses CPU resources to:

- Send and receive messages
- Maintain locks and resources required to guarantee global cache coherency
- Process additional I/O requests due to the invalidation of cache buffers associated with ping processing

An additional cost is incurred waiting for I/O events and lock open or convert requests. Conflicts for resources between local and remote transactions while opening or converting a lock on a database resource can increase the cost of synchronization.

As a general example, approximate the cost of a transaction or request using this formula:

$$CPUapps + CPUsyncio + CPUipc + CPUlocks + WAITsyncio + WAITipc + WAITlocks$$

You can also calculate:

- CPU time spent at the IPC layer
- CPU time required to process lock requests
- CPU spent for application processing, such as parsing SQL statements, fetching rows, and sorting
- CPU time spent in processing read and write I/O caused by synchronization between nodes

Note: Delays associated with these events are called "waits".

Statistics about these events appear in V$ tables such as V$SYSTEM_EVENT. The response time for each transaction depends on the number of requests for I/O and locks, and on the time required to process the instructions, plus the delay or wait time for each request.

Contention on certain resources adds to the cost of each measurable component. For example, increased disk service or queueing times due to:

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

can result in waits for busy buffers. This can in turn increase costs for each transaction and add to the operating system overhead.

As described in Chapter 5, you can create Oracle Parallel Server applications that are more scalable and that perform well enough to meet the service level requirements of a user community by minimizing either the rate or the cost of synchronization. In other words, strive to minimize inter-node synchronization and communication requirements. Partitioning, for example, where you isolate portions of an application that you can process by a group of transactions without interference from other transactions, reduces the cost and rate of global synchronization.

The choice of locking mechanisms and lock allocation, as described in Chapter 7, also has an effect on the rate and cost of global cache coherence. Recent optimizations in Oracle Parallel Server, such as Cache Fusion, attempt to reduce the cost of operations across instances by decreasing the delays for requests for shared or exclusive buffer access.

Determining the Costs of Synchronization

This section explains how to determine the cost incurred by synchronization and coherence between instances due to additional CPU time, I/O and global lock processing and contention. To do this, examine these groups of Oracle statistics:

- Calculating CPU Service Time Required
- Estimating I/O Synchronization Costs
- Measuring Global Cache Coherence and Contention
- Measuring Global and Local Work Ratios
- Calculating the Cost of Global Cache Synchronization Due to Lock Contention

Calculating CPU Service Time Required

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

This measure is useful for comparing how applications behave in single instance environments running in exclusive mode to applications running in an Oracle Parallel Server environment. This measure is also instrumental in comparing the effect of different workloads and application design changes.

Estimating I/O Synchronization Costs

Refer to the V$SYSSTAT view for counts of the following requests:

- DBWR cross-instance writes
- physical writes
- physical reads

Refer to the V$SYSTEM_EVENT view for time waited and average waits for the following actions:

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

To estimate the time waited for reads incurred by rereading data blocks that had to be written to disk because of a request from another instance, multiply the statistic (for example, the time waited for db file sequential reads) by the percentage of read I/O caused by previous cache flushes as shown in this formula:

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

Where "lock buffers for read" is the value for lock converts from N to S derived from V$LOCK_ACTIVITY and "physical reads" is from the V$SYSSTAT view.

Similarly, the proportion of the time waited for database file parallel writes caused by pings can be estimated by multiplying db file parallel write time as found in V$SYSTEM_EVENTS by:

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

Measuring Global Cache Coherence and Contention

Table 11–1 describes some global cache coherence-related views and the types of statistics they contain.

Table 11–1 Global Cache Coherence and Contention Statistics and Their Views

View	Statistics
Refer to V$SYSSTAT to count requests for the actions shown to the right.	global cache gets (count of new locks opened) global cache converts (count of conversion for existing locks) global cache cr blocks received (count of consistent read buffers received from the BSP) Note: Also refer to the convert type-specific rows in V$LOCK_ACTIVITY.
Refer to V$SYSSTAT for the amount of time waited for the actions shown to the right.	global cache get time (total processing time including waits) global cache convert time (total processing time including waits) global cache cr block receive time (includes waits)
Refer to V$SYSTEM_EVENT for time waited for the events shown to the right.	global cache lock null to X global cache lock null to S global cache lock S to X global cache lock open X global cache lock open S global cache cr request

For indicators of high contention or excessive delays, refer to the following statistics and views:

- global cache cr timeouts as found in V$SYSSTAT
- global cache convert timeouts as found in V$SYSSTAT

- global cache lock busy as found in V$SYSTEM_EVENT
- buffer busy due to global cache as found in V$SYSTEM_EVENT

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

- lock gets per transaction, for example, global cache gets per transaction
- lock converts per transaction, for example, global cache converts per transaction
- cr request per transaction, for example, global cache cr blocks received per transaction
- lock convert waits per transaction, the value shown in the TIME_WAITED column in V$SYSTEM_EVENT
- global cache lock null to x per transaction
- global cache lock null to s per transaction
- global cache lock s to x per transaction
- lock open waits per transaction, for example, from the TIME_WAITED column in V$SYSTEM_EVENT
- global cache lock open x per transaction
- global cache lock open per transaction
- lock busy waits per transaction, for example, from the TIME_WAITED column in V$SYSTEM_EVENT
- global cache lock busy per transaction

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

Note: To record timed statistics, set the TIMED_STATISTICS parameter to TRUE. Oracle records these statistics in hundredths of seconds.

Measuring Global and Local Work Ratios

The percentage of buffers accessed for global work or the percentage of I/O caused by inter-instance synchronization can be important measures of how efficient your application processes share data. It can also reveal whether the database is designed for optimum scalability.

Use the following calculation to determine the percentage of buffer accesses for local operations, in other words, reads and changes of database buffers that are not subject to a lock conversion:

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

Similarly, compute the percentage of read and write I/O for local operations using the following equations:

$$\frac{(\textit{physical writes} - (\textit{DBWR cross-instance writes})) * 100}{\textit{physical writes}}$$

This calculation implies the percent of times DBWR writes for local work.

Also:

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

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

In the previous formula, the physical read statistic from V$SYSSTAT is combined with the "Lock buffers for read" value from V$LOCK_ACTIVITY. You can base the local write ratio entirely on the corresponding values from V$SYSSTAT.

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

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

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

More generally:

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

For lock opens and converts, the percentage of waits that are caused by busy locks (locks that are being acquired or released) can be indicative of either delays in opening or converting locks or very high concurrency on a small set of buffers.

$$\frac{100 * (\mathit{time\ waited\ for\ global\ cache\ lock\ busy})}{\mathit{sum}(\mathit{time\ waited\ for\ global\ cache\ opens\ and\ global\ cache\ converts})}$$

Once you identify a problem area, such as a high ratio of global busy waits, convert and consistent read timeouts, or a high percentage of DBWR cross-instance writes, you can obtain more detail about the problem by referring to these views:

- V$FILE_PING
- V$CACHE
- V$CLASS_PING

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

Calculating the Cost of Global Cache Synchronization Due to Lock Contention

Reduced throughput and degradation of transaction response times are the result of the increased costs of synchronization between instances. There are several possible sources of such increased costs as described in this section under the following headings:

- Contention for the Same Data Blocks
- Contention for Segment Headers and Free List Blocks
- Contention for Resources Other Than Database Blocks
- A Shortage of Locks

Contention for the Same Data Blocks

Contention for the same data blocks occurs if rows commonly accessed from multiple instances are spread over a limited range of blocks. The probability of this happening depends on the access distribution of the data within the table as a result of the application's behavior. The probability of contention can also depend on the block size. For example, more rows fit into an 8K block than into a 4K block. The PCTFREE defined for that particular table can also affect the level of contention. In fact, database block size and PCTFREE can be part of your Oracle Parallel Server design strategy: your goal is to reduce the number of rows per block and thus the probability of conflicting access.

Indicators of very "hot" globally accessed blocks include:

- A high percentage of buffer busy waits or buffer busy due to global cache waits
- Frequent convert timeouts or consistent read timeouts

If you see a high proportion of global cache lock waits per transaction, consider determining which files and blocks are accessed frequently from both nodes. The following describes one method for identifying contended files.

Using V$CACHE, V$PING, and V$BH to Identify Contended Objects

The dynamic performance views V$CACHE, V$PING and V$BH have two columns, FORCED_READS and FORCED_WRITES, that allow you to determine which objects and blocks are used by both nodes.

The FORCED_WRITE column carries a count of how many times a certain block was pinged out of the local buffer cache, that is, written to disk because the current version was requested by another instance. Correspondingly, the FORCED_READ column tracks how frequently a particular block had to be reread from disk because it was previously invalidated by an exclusive request from another instance. You can also use V$FILE_PING to identify files that experience the most pinging.

Using V$FILE_PING to Identify Files with Excessive Pinging

Use the V$FILE_PING view to identify files containing blocks that are most frequently written to due to pings. Once you have identified the files with the highest ping rates, reconsider your locking policy for these files (fixed or releasable). You may also want to consider whether you should physically distribute these files to reduce I/O delays. Of course, the best strategy is to increase the percentage of local work on the objects contained in the files, that is, to avoid pinging altogether.

Note: A frequently pinged block will be visible in the local buffer caches of all instances in the cluster.

Contention for Segment Headers and Free List Blocks

Contention for segment headers can occur when the headers for tables or indexes have to be read and updated by many transactions at the same time. Usually this happens when transactions are searching for blocks with enough free space to hold the size of the data to be inserted or updated. Oracle also updates a segment header if new extents or additional free blocks are added to the table.

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

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

Contention for Resources Other Than Database Blocks

Contention for resources other than database blocks should be infrequent. However, when this occurs, it will definitely have an adverse affect on performance. Usually, there are two layers that may exhibit such problems:

- The row cache or data dictionary cache
- The library cache

Using Oracle sequences that are not cached, or using poorly configured space parameters for a table or tablespace, may result in frequent changes to the data dictionary. Rows from the data dictionary are stored in a separate cache and use a different buffer format than that of the data buffer cache. If these objects are often read and updated from both nodes, Oracle must invalidate them and write them to disk. Unfortunately, these objects are often used in recursive, internal transactions while other locks are held. Because of the complexity of such internal data dictionary transactions, this processing can cause serious delays.

When this occurs, the values for "row cache lock" wait count and "time waited" statistics in V$SYSTEM_EVENT increase; these become two of the most waited-for events. The percentage of time waited for row cache locks should never be more than 5% of the total wait time. To determine which objects in the row cache may be causing the delays, query the V$ROWCACHE view. Oracle's response to the query is the name of the row cache object type, such as DC_SEQUENCES or DC_USED EXTENTS, and the DLM requests and DLM conflicts for these objects. If the conflicts exceed 10 to 15 percent of the requests and row cache lock wait time is excessive, then take preventive action by caching sequence numbers, defragmenting tablespaces, or by tuning space parameters.

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

If there are frequent changes to the data dictionary, many data blocks from the data dictionary tables, normally from file number 1, can be found in each instance's buffer cache when you query V$CACHE or V$PING.

Library cache performance problems in Oracle Parallel Server are rare. They usually manifest themselves as high wait times for "library cache pin" and could be caused by the frequent reparsing of cursors or by the loading of procedures and packages. Query the DLM column in the V$LIBRARYCACHE view to obtain more detail about this problem. Cross-instance invalidations of library cache objects should be rare. However, such invalidations can occur if you drop objects referenced by a cursor that is executed in two instances.

A Shortage of Locks

A shortage of locks or resources may occur if the database and the buffer cache are large and if there is a constraint on memory. You can set the parameter GC_RELEASABLE_LOCKS to a value lower than the default. This value should be equal to the size of the buffer cache as determined by DB_BLOCK_BUFFERS. This allows you to save memory for locks and resources can be saved.

However, if a lot of resources are used to maintain global cache coherency and the free lists are depleted, locks need to be freed and reopened more frequently. An increasing count of global cache lock free list waits and global cache gets or global cache lock open events can indicate this. Waiting for locks to be freed or a higher percentage of locks that are opened add to the cost incurred by synchronizing instances and usually results in higher transaction response times and higher CPU use.

Resolving Problems in Oracle Parallel Server-Based Applications

This section explains how to identify and resolve problems in Oracle Parallel Server-based applications. It contains the following sections:

- Query Tuning Tips
- Query Tuning Tips
- Application Tuning Tips
- Contention Problems Specific to Parallel Server Environments

Query Tuning Tips

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

Large Block Size

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

Increase Value for DB_FILE_MULTIBLOCK_READ_COUNT

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

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

Also remember to:

- Use read-only tablespaces for data and indexes
- Define tablespaces holding temporary segments as type TEMPORARY and use a large blocking factor for PCM locks on data files containing tables that are frequently fully scanned

Application Tuning Tips

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

To improve the ability of the database writer processes (DBWn) to write large amounts of data quickly, use asynchronous I/O. Oracle8*i* uses multiple DBWn processes to improve performance. You can also use low granularity locking to avoid false pings.

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

- Be aware of contention on related indexes and sequence generators
- Consider using a multi-tiered architecture to route users for partitioning and for failover

Diagnosing Performance Problems

If your application is not performing well, analyze each component of the application to identify which components are causing problems. To do this, check the operating system and DLM statistics, as explained under the next heading, for indications of contention or excessive CPU usage. Excessive lock conversions that you can measure with specific procedures may reveal excessive read / write activity or high CPU requirements by DLM components.

See Also: Chapter 11 for more information about tuning Oracle Parallel Server performance.

DLM Statistics for Monitoring Contention and CPU Usage

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

- Use standard tuning techniques by running UTLBSTAT, UTLESTAT, and then querying the V$SQL_AREA view. Examine the statistics from this view and analyze the hit ratios in the shared pool and the buffer cache.
- Examine the dynamic performance table statistics created when you run CATPARR.SQL.
- Use the V$LOCK_ACTIVITY table to monitor lock rates.
- Use the V$BH table to identify which blocks are being pinged. This table sums the number of times each block's PCM locks are downgraded from exclusive to null.
- The V$PING view shows rows from the V$CACHE table where the exclusive to null count is non-zero.

Note: Blocks that are newly acquired by an object do not appear in these tables until you rerun CATPARR.SQL.

Contention Problems Specific to Parallel Server Environments

There are significant contention problems that you can avoid in Oracle Parallel Server environments. These are the result of inserts into index blocks when multiple instances share a sequence generator for primary key values. You can minimize these problems by:

- Using Sequence Number Multipliers
- Using Oracle Sequences

Using Sequence Number Multipliers

You may need to use a multiplier such as SEQUENCE_NUMBER x INSTANCE_NUMBER x 1,000,000,000 to prevent the instances from inserting new entries into the same index.

Using Oracle Sequences

Creating a sequence without using the CACHE clause may create a lot of overhead. It may also cause synchronization overhead if both instances use the same sequence.

12

Tuning Oracle Parallel Server and Inter-Instance Performance

The chapter describes Oracle Parallel Server and Cache Fusion-related statistics and provides procedures that explain how to use these statistics to monitor and tune performance. This chapter also briefly explains how Cache Fusion resolves reader / writer conflicts in Oracle Parallel Server. It describes Cache Fusion's benefits in general terms that apply to most types of systems and applications.

The topics in this chapter include:

- How Cache Fusion Produces Consistent Read Blocks
- Improved Scalability with Cache Fusion
- The Interconnect and Interconnect Protocols for Oracle Parallel Server
- Performance Expectations
- Monitoring Cache Fusion and Inter-Instance Performance

See Also: *Oracle8i Parallel Server Concepts* for an overview of Cache Fusion processing.

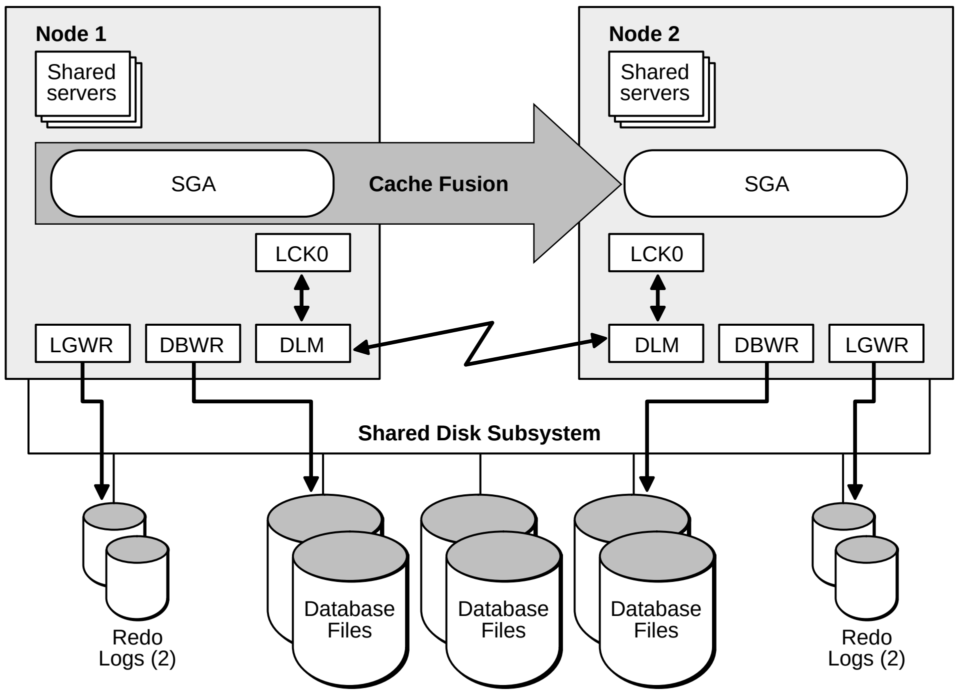
How Cache Fusion Produces Consistent Read Blocks

When a data block requested by one instance is in the memory cache of a remote instance, Cache Fusion resolves the read / write 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 Block Server Process (BSP) 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 12–1 illustrates, Cache Fusion enables the buffer cache of one node to send 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 potentially lowers CPU usage by reducing operating system context switches for inter-node messages. Oracle manages write / write contention using conventional disk-based Parallel Cache Management (PCM).

Figure 12–1 Cache Fusion Ships Blocks from Cache to Cache Across the Interconnect



Note: Cache Fusion is always enabled.

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 concurrency, you also need to accurately partition your application's tables to reduce the potential for writer/writer conflicts.

Improved Scalability with Cache Fusion

Cache Fusion improves application transaction throughput and scalability by providing:

- Reduced 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. 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.

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

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.

Consistent-Read Block Transfers By Way of High Speed Interconnects

Because Cache Fusion exploits high speed IPCs, Oracle Parallel Server 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 Virtual Interface Architecture (VIA) and 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, if efficiently implemented by hardware support, can reduce CPU use 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 percent.

Cache Fusion resolves reader/writer concurrency with approximately one tenth of 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 one millisecond.

The Interconnect and Interconnect Protocols for Oracle Parallel Server

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 that support Oracle Parallel Server and Cache Fusion use one of these protocols:

- TCP/IP (Transmission Control Protocol/Interconnect Protocol)
- UDP (User Datagram Protocol)
- VIA (Virtual Interface Architecture)
- Other proprietary protocols that are hardware vendor-specific

Oracle Parallel Server can use any interconnect product that supports these protocols. The interconnect product must also be certified for Oracle Parallel Server 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.

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 significantly.

A comparison of the locking and I/O statistics for Oracle 8.1. and Oracle 8.0 reveals a major reduction of exclusive to shared lock requests and physical write I/O. The following section, "Monitoring Cache Fusion and Inter-Instance Performance", describes how to evaluate Cache Fusion performance in more detail.

See Also: Chapter 7 for more information on lock types.

Monitoring Cache Fusion and Inter-Instance Performance

This section describes how to obtain and analyze Oracle Parallel Server and Cache Fusion statistics to monitor inter-instance performance. Topics in this section include:

- Cache Fusion and Oracle Parallel Server Performance Monitoring Goals
- Statistics for Monitoring Oracle Parallel Server and Cache Fusion
- Using the V$SYSTEM_EVENTS View to Identify Performance Problems

Cache Fusion and Oracle Parallel Server Performance Monitoring Goals

The main goal of monitoring Cache Fusion and Oracle Parallel Server performance is to determine the cost of global processing and quantify the resources required to maintain coherency and synchronize the instances. Do this by analyzing the performance statistics from several views as described in the following sections. Use these monitoring procedures on an ongoing basis to observe processing trends and to maintain processing at optimal levels.

Many statistics are available to measure the work done by different components of the database kernel, such as the cache layer, the transaction layer or the I/O layer. Moreover, timed statistics allow you to accurately determine the time spent on processing certain requests or the time waited for specific events.

From these statistics sources, work rates, wait time and efficiency ratios can be derived.

See Also: Chapter 11 for additional suggestions on which statistics to collect and how to use them to compute performance ratios.

Statistics for Monitoring Oracle Parallel Server and Cache Fusion

Oracle collects Cache Fusion-related performance statistics from the buffer cache and DLM layers. Oracle also collects general Oracle Parallel Server 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 identify trends as these statistics change. This facilitates identifying contributors to increasing response times and reduced throughput. It would also be helpful in spotting workload changes and peak processing requirements.

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 DLM Resource, Lock, Message, and Memory Resource Statistics
- DLM Message Statistics
- Analyzing Oracle Parallel Server I/O Statistics
- Analyzing Latch, Oracle Parallel Server, and DLM Statistics

As described in Chapter 11, consider maintaining statistics from the V$SYSSTAT view and the V$SYSTEM_EVENT view on a per second and per transaction basis to obtain a general profile of the workload. Relevant observations from these views are:

- Requests or counts per transaction, for example, physical reads per transaction and logical reads per transaction
- Wait times or elapsed time per transaction, for example, read I/O wait time per transaction and lock convert time per transaction
- Requests or counts per second
- Average times per request, for example, average time to receive a consistent read buffer from another instance

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

You must set the parameter TIMED_STATISTICS to TRUE 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 Oracle Parallel Server-related views and tables for storing and viewing statistics as described under the next heading.

Creating Oracle Parallel Server 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 well as:

- GV$CACHE
- GV$PING
- GV$CLASS_PING
- GV$FILE_PING
- GV$ROWCACHE
- GV$LIBRARYCACHE

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.

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 Oracle Parallel Server, 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.

See Also:

- Chapter 2
- *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 DLM, Oracle allocates data structures in the System Global Area 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 cr blocks received                                    7372
global cache cr block receive time                                 2293
global cache cr blocks served                                      7882
global cache cr block serve time                                     60
global cache cr block send time                                     239
global cache cr block log flushes                                   119
global cache cr block log flush time                                140
global cache cr timeouts                                              2
global cache cr requests blocked                                      0
```

2. The average latency of a consistent block request, in other words, its round-trip time, can be calculated as:

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

   The result, which should typically be about 15 milliseconds depending on your system configuration and volume, is the average latency of a consistent-read request round trip from 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 one millisecond.

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. For a high number of incoming requests, especially in report-intensive applications, or if there are multiple nodes from which requests can arrive at a BSP, the round-trip time can increase because BSP's service time increases. To determine whether the length of the delay is caused by BSP, compute the average service time per request.

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

Track the average BSP service time per request and the total round-trip time per request as presented in this step.

To determine which part of the service time correlates most with the total service time, derive the time waited for a log flush and the time spent in sending the completed request using the following two equations:

$$\frac{\textit{global cache cr log flush time}}{\textit{global cache cr log flushes}}$$

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

By calculating these averages, you can account for almost all the processing steps of a consistent read block request. The remaining difference between the total round-trip time and the BSP service time per request falls onto processing time in the LMD processing and network IPC time.

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

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

   High convert times may indicate excessive global concurrency. A large number of global cache gets, global cache converts, and a rapid increase in average convert or get times indicates that 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 get is 20 to 30 milliseconds while converts should take 10 to 20 milliseconds 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 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, 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. It might also be used as an indicator of IPC or network problems.

Other Useful Cache Fusion Statistics

The following describes additional Cache Fusion statistics that you may find useful in diagnosing global cache and Cache Fusion operations. Use these statistics to monitor all the major operations of a consistent block request.

global cache cr blocks received When a process requests a consistent read for a data block that it cannot satisfy from its local cache, it sends a request to another instance. Once the request is complete, in other words, the buffer has been received, Oracle decrements the request count.

global cache cr block receive time This statistic records the total time it took for consistent read requests to complete, in other words, the accumulated round-trip time for all requests for consistent read blocks.

global cache cr timeouts This statistic identifies a request for a consistent read block that has a long delay and that has timed out. This could be due to system performance problems, a slow interconnect network or dropped network packets. The value of this statistic should always be 0.

global cache cr blocks served This is the number of requests for a consistent read block served by BSP. Oracle increments this statistic when the block is sent.

global cache cr block serve time This statistic represents the accumulated time it took BSP to fill all incoming requests. For each request, the start time is recorded immediately after BSP takes a request off the request queue. The interval is computed after the blocks is sent.

global cache cr block send time This is the time required by BSP to initiate a send of a consistent read block. For each request, timing starts when the block is sent and stops when the send has completed. It is a part of the serve time. Note that this statistic only measures the time it takes to initiate the send; it does not measure the time elapsed before the block arrives at the requestor.

global cache cr block log flushes For changes to buffers containing a version of a data block that the block sever process has produced, a log flush must be initiated. BSP handles the wait asynchronously by managing a completion queue. Once LGWR has completed flushing the changes to a buffer that is on the log flush queue, BSP can send it. Therefore it periodically checks the queue. Oracle increments this statistic when a log flush is queued.

global cache cr block log flush time This is the time waited for a log flush. It is part of the serve time.

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 DLM API layer. All Oracle clients to the DLM, of which the buffer cache is only one, make their requests to the DLM through this layer. Thus, global lock statistics include lock requests originating from all layers of the kernel, while global cache statistics relate to buffer cache Oracle Parallel Server activity.

Use the procedures in this section to monitor data from the V$SYSSTAT view to derive averages, latencies, and counts. This establishes a rough indicator of the Oracle Parallel Server workload generated by an instance.

Procedures for Analyzing Global Lock Statistics

Use the following procedures to view and analyze statistics from the V$SYSSTAT view 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 group of procedures.

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 30 milliseconds, query the TIME_WAITED column in the V$SYSTEM_EVENTS view 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 20 milliseconds, query the TIME_WAITED column in the V$SYSTEM_EVENTS view using the DESCEND keyword to identify the event causing the delay.

4. As mentioned, global lock statistics apply to buffer cache lock operations and lock operations for resources other than data blocks. To determine which type of resources may be a performance problem, divide the global lock get and global lock convert statistics into two categories:

– Synchronous Operations

  Synchronous lock gets includes, for example, global lock sync gets. These are usually performed for lock requests for resources other than cached data blocks. To determine the proportion of the time required for synchronous lock gets, divide global lock get time or global lock convert time by the corresponding number of synchronous operations.

– Asynchronous Operations

  Asynchronous lock operations include, for example, global lock async gets. These are typically lock operations for global cache locks. You can derive the proportion of the total time using the same calculation as for synchronous operations. In this way, the proportion of work and the cost of global cache lock requests and other lock requests can be determined.

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

5. Analyze the V$LIBRARYCACHE and V$ROWCACHE views to observe DLM activity on non-PCM resources. These views have DLM-specific columns that identify DLM resource use. Analyze these views for DLM activity if you have frequent and extended waits for library cache pins, enqueues, or DFS lock handles.

Analyzing DLM Resource, Lock, Message, and Memory Resource Statistics

Oracle collects DLM resource, lock, and message statistics at the DLM level. Use these statistics to monitor DLM 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. The V$DLM_CONVERT_LOCAL view shows statistics for local lock operations. The V$DLM_CONVERT_REMOTE view 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 DLM Workloads Affect Performance

The DLM workload is an important aspect of Oracle Parallel Server and Cache Fusion performance because each consistent-read request results in a lock request. High DLM workloads as a result of heavy request rates can adversely affect performance.

The DLM 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 DLM.

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

Procedures for Analyzing DLM Resource and Lock Statistics

Use the following procedures to obtain and analyze statistics from the V$DLM_CONVERT_LOCAL and V$DLM_CONVERT_REMOTE views 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 the V$DLM_CONVERT_LOCAL view:

   ```
   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 the V$DLM_CONVERT_REMOTE view:

   ```
   SELECT * FROM V$DLM_CONVERT_REMOTE;
   ```

   Oracle responds with output identical in format to the output for the V$DLM_CONVERT_LOCAL view.

   Use your output from the V$DLM_CONVERT_LOCAL and V$DLM_CONVERT_REMOTE views 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 lock requests per transaction without increases in average latencies usually result 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 is greater than 20 percent 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" on UNIX or Perfmon on Windows NT.

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

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

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

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

DLM Message Statistics

The DLM sends messages either directly or by using flow control. For both methods, the DLM attaches markers known as "tickets" to each message. The allotment of tickets for each DLM is limited. However, the DLM can re-use tickets indefinitely.

DLMs send messages directly until no more tickets are available. When an DLM runs out of tickets, messages must wait in a flow control queue until outstanding messages have been acknowledged 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 DLM Message Statistics

Use the following procedures to obtain and analyze message statistics in the V$DLM_MISC view.

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

```
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 the V$DLM_MISC view 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 Oracle Parallel Server I/O Statistics

In addition to the global cache and global lock statistics that were previously discussed, you can also use statistics in the V$SYSSTAT view to measure the I/O workload related to global cache synchronization. There are three important statistics in the V$SYSSTAT view 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.

Cache Fusion minimizes the disk I/O for consistent-reads. This can lead to a substantial reduction in physical writes and reads performed by each instance. Before Cache Fusion, a consistent-read requesting data from a remote instance could result in up to three write I/Os on the remote instance and three corresponding read I/Os for the requesting instance: one for the data block, one for the rollback segment header, and one for a rollback segment block.

Analyzing Oracle Parallel Server I/O Statistics in the V$SYSSTAT View

You can obtain the following statistics to quantify the write I/Os required for global cache synchronization.

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

   ```
   SELECT NAME, VALUE FROM V$SYSSTAT
   WHERE NAME IN ('DBWR 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 cross-instance writes                                    5403
   remote instance undo block writes                                0
   remote instance undo header writes                               2
   4 rows selected.
   ```

   Where the statistic "physical writes" refers to all physical writes that occurred from a particular instance performed by DBWR, the value for "DBWR cross-instance writes" accounts for all writes caused by writing a dirty buffer containing a data block that is requested for modification by another instance. As cross-instance writes are also handled by DBWR, it follows that "DBWR cross-instance writes" is a subset of all "physical writes".

   The other notable statistics, "remote instance undo block writes" and "remote instance undo header writes", refer to the number of times that Oracle writes a rollback segment block to disk because another instance intends to build a consistent read version of a data block but the information required to roll back the block are not in the instance's cache. Both are a subset of "DBWR cross-instance writes". Their significance for performance is less critical in Oracle8*i* because Cache Fusion reduces the need to "export" and "import" rollback information. In most cases, instances send the complete version of a data block by way of the interconnect to the requesting instance.

   Note that every lock conversion from Exclusive (X) to Null (N) or from Exclusive (X) to Shared (S) is associated with a write to disk when the buffer under the lock is dirty. However, in Oracle8*i*, the number of X to S lock conversions is reduced because Cache Fusion does not require them. In most cases, the holding instance retains the X lock.

2. Calculate the ratio of Oracle Parallel Server-related I/O to overall physical I/O using this equation:

$$\frac{\textit{DBWR forced writes}}{\textit{physical writes}}$$

   You should see a noticeable decrease in this ratio between this calculation and pre-Cache Fusion statistics.

3. Use this equation to calculate how many writes to rollback segments occur when a remote instance needs to read from rollback segments that are in use by a local instance:

$$\frac{\begin{array}{c}\textit{(remote instance undo header writes+}\\ \textit{remote instance undo block writes)}\end{array}}{\textit{DBWR forced writes}}$$

   The ratio shows how much disk I/O is related to writes to rollback segments. With Cache Fusion, this ratio should be very low.

4. To estimate the number or percentage of reads due to global cache synchronization, use the number of lock requests for conversions from NULL(N) to Shared mode (S) counted in V$LOCK_ACTIVITY and the "physical reads" statistics from V$SYSSTAT.

   The following formula computes the percentage of reads that are only for local work:

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

   Where "lock buffers for read" represents the N to S lock conversions.

   These so-called "forced reads" occur when a cached data block that was previously modified by the local instance had to be written to disk due to a "ping" from another instance and the block is then re-acquired by the local instance for a read.

See Also: Chapter 11 for more observations regarding estimations of local and global work rates and percentages in Oracle Parallel Server clusters.

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 the V$LOCK_ACTIVITY View to Analyze Lock Conversions
- Using the V$CLASS_PING View to Identify Pinging by Block Class
- Using the V$PING View to Identify Hot Objects

Using the V$LOCK_ACTIVITY View to Analyze Lock Conversions

The V$LOCK_ACTIVITY view 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 the V$CLASS_PING View to Identify Pinging by Block Class

The V$CLASS_PING view 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 the V$PING View to Identify Hot Objects

The V$PING view helps identify "hot" blocks and "hot" objects. The sum of each column, FORCED_READS and FORCED_WRITES, indicates the actual pinging activity on a particular block or object.

All three views provide different levels of detail. If you suspect that pinging or Oracle Parallel Server itself is the cause of a performance problem, monitor the V$LOCK_ACTIVITY view to generate an overall Oracle Parallel Server workload profile. Use information from the V$LOCK_ACTIVITY view to record the rate at which lock conversions occur.

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

If your response time or throughput requirements are no longer being met, you would normally examine the V$LOCK_ACTIVITY, V$CLASS_PING, V$CACHE, V$PING or V$FILE_PING views. In addition, you might also examine:

- V$SYSSTAT to identify an increase in lock requests per transaction
- V$SYSSTEM_EVENT to identify longer waits for global cache locks or consistent read server requests per transaction
- Global and local work done as described in Chapter 11 to see if there is a noticeable change in performance percentages

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

Analyzing Latch, Oracle Parallel Server, and DLM Statistics

Latches are low-level locking mechanisms that protect System Global Area data structures. Excessive contention for latches degrades performance.

Use the V$DLM_LATCH and V$LATCH_MISSES views to monitor latch contention within the DLM. These views show information about a particular latch, its statistics, and the location in the code from where the latch is acquired.

For normal operations, the value latch statistics is limited. In some cases, multiple latches can help increase the performance for certain layers by a small amount. High latch contention is often the result of either:

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

The following procedures are suggestions as to which information is available. Oracle does not recommend that you monitor these statistics on a regular basis and derive conclusions solely on the basis of latching issues. However, gathering this information might be useful to Oracle Support or Oracle's Development Staff. Also, latch tuning can be the object of advanced tuning activities, but in the majority of cases latch tuning will not be your actual performance problem.

On the other hand, record information from these procedures if the TIME_WAITED value for the "latch free" wait event is very high and ranks among the events that accrue the largest times as indicated by the V$SYSTEM_EVENT view.

Procedures for Analyzing Latch, Parallel Server, and DLM Statistics

Use the following procedures to analyze latch, Oracle Parallel Server, and DLM-related statistics.

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

   ```
   SELECT * FROM V$LATCH;
   ```

   Oracle responds with output similar to the following where the columns from left to right show the statistic name, gets, misses, and sleeps:

   ```
   cache buffer handle              46184        1        0
   cache buffers chained         84139946   296547    29899
   cache buffers lru              4760378    11718      227
   channel handle pool                  1        0        0
   channel operations                   1        0        0
   dlm ast latch                   542776      494     1658
   ```

```
dlm cr bast queue                37194          1          0
dlm deadlock list                32839          0          0
dlm domain lock la                   1          0          0
dlm domain lock ta               49164          1          0
dlm group lock lat                   1          0          0
dlm group lock tab               25239          1          0
dlm lock table fre              325306        270        327
dlm process hash l                6346          0          0
dlm process table                    2          0          0
dlm recovery domai                2014          0          0
dlm resource hash               683031       1709      41342
dlm resource scan                  188          0          0
dlm resource table              182093         70          2
dlm shared communication        190766        211        313
dlm timeout list                113294         40          3
dml lock allocation                261          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 step 1 in this section 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 step 1 in this section. 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 the V$LATCH_MISSES view, the WHERE column shows the function in which the latch is acquired. This information is useful in determining internal performance problems. Usually, the latch slept on for long periods shows up in the V$SESSION_WAIT or V$SYSTEM_EVENT views under the 'latch free' wait event category.

The following section describes how to use the V$SYSTEM_EVENTS view in more detail.

Using the V$SYSTEM_EVENTS View to Identify Performance Problems

Data about Cache Fusion and Oracle Parallel Server events appears in the V$SYSTEM_EVENT view. To identify events for which processes have waited the longest, query the V$SYSTEM_EVENT view on the TIME_WAITED column using the DESCENDING keyword. The TIME_WAITED column shows the total wait time for each system event listed.

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.

Oracle recommends dividing the sum of values from the TOTAL_WAITS and TIME_WAITED columns by the number of transactions, as outlined in Chapter 11. Transactions can be defined as business transactions, for example, insurance quotes, order entry, and so on, or you can define them on the basis of "user commits" or "executions", depending on your perspective.

The goal is to estimate which event type contributes primarily to transaction response times, since in general:

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

By this rationale, the total wait time can be divided into subcomponents of the wait time, such as:

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

where *tm* is "time waited".

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

Parallel Server Events in V$SYSTEM_EVENTS

The following events appearing in the V$SYSTEM_EVENT output represent waits for Oracle Parallel Server 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 Oracle Parallel Server. 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 Oracle Parallel Server-related issues. Specifically, observe values in the DLM columns of each of these views.

Common Oracle Parallel Server 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 the V$ROWCACHE view will show a high number of conflicts for certain dictionary caches.

Part V

Oracle Parallel Server Maintenance

Part Five provides information about backup and recovery by presenting the following chapters:

13
Backing Up Your Database

To protect your data, archive the online redo log files and periodically back up the data files. 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 the following topics:

Oracle Parallel Server supports all Oracle backup features in exclusive mode, including both open and closed backup of either an entire database or individual tablespaces.

See Also:

- *Oracle8i Backup and Recovery Guide* for general information about backup and recovery.
- *Oracle8i Recovery Manager User's Guide and Reference* for information about Recovery Manager (RMAN).

Choosing a Backup Method

You can perform backup and recovery operations using two methods:

- Using Recovery Manager (RMAN)
- Using the operating system

The information provided in this chapter is useful for both methods, unless specified otherwise.

See Also: *Oracle Enterprise Manager Administrator's Guide* about using Oracle Enterprise Manager's Backup Wizard.

To avoid confusion between online and offline data files 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 data files and control files have been backed up. "Full" and "incremental" backups refer only to particular types of backups provided by RMAN.

See Also: *Oracle8i Recovery Manager User's Guide and Reference* for a complete discussion of backup and recovery operations and terminology related to RMAN.

Archiving the Redo Log Files

This section explains how to archive the redo log files for each instance of Oracle 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. 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 failure. In ARCHIVELOG mode, you can produce both open and closed backups. In NOARCHIVELOG mode, you can make only closed backups.

Note: Archiving is a per-instance operation that can be handled in one of two ways:

- Each instance on Oracle 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 13-16.

Changing the Archiving Mode

Determine whether to use archive logging which preserves groups of online redo log files. Without archive logging, Oracle overwrites redo log files once they are available for reuse.

The choice of whether to enable the archiving of filled online redo log files depends on your application's availability and reliability requirements. If you cannot afford to lose any data in the event of a disk failure, use ARCHIVELOG mode. Note that archiving filled online redo log files can require extra administrative operations.

See Also: *Oracle8i Parallel Server Setup and Configuration Guide* for information on how to configure archive logs in Oracle Parallel Server.

To enable archive logging in Oracle Parallel Server environments, the database must be mounted but not open. Then start Parallel Server in a disabled state. To do this:

1. Shut down all instances.
2. Reset the parameter PARALLEL_SERVER to FALSE on one instance.

3. Start up the instance on which you have set PARALLEL_SERVER to FALSE.
4. Enter the following statement:

   ```
   ALTER DATABASE ARCHIVELOG
   ```

5. Shut down the instance.
6. Change the value of the PARALLEL_SERVER parameter to TRUE.
7. Restart your instances.

To disable archive logging, follow the same steps but use the NOARCHIVELOG clause of the ALTER DATABASE statement.

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 Oracle Parallel Server instance. For example, you can manually use SQL statements 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 statement ALTER SYSTEM.
- Enable automatic archiving using the SQL statement ALTER SYSTEM ARCHIVE LOG START.

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 Clauses for Manual Archiving

ALTER SYSTEM ARCHIVE LOG manual archiving clauses 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 clause 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 13-15 regarding threads and log switches.
- *Oracle8i Reference* for information about the syntax of the ALTER SYSTEM ARCHIVE LOG statement.
- *Oracle8i Recovery Manager User's Guide and Reference* as well as the "Managing 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:** *Oracle8i Reference* for more information about these views.

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 13–1. Examples in this table assume that LOG_ARCHIVE_FORMAT= arch%*parameter*, and the upper bound for all parameters is 10 characters.

Table 13–1 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 Oracle Parallel Server. 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:

- *Oracle8i Administrator's Guide* for information about specifying the archived redo log filename format and destination.
- Oracle system-specific documentation for 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 statement to make the control file retain a history of redo log files that an instance has filled. After creating the database, you can only increase or decrease the log history by creating new control files. Using CREATE CONTROLFILE destroys all log history in the current control file.

The MAXLOGHISTORY clause 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, SQL*Plus prompts you for the appropriate file names. RMAN 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$LOG_HISTORY view. V$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 14-8 for SQL*Plus 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 accessible only by the node on which they were created. In Oracle Parallel Server you have three backup options:

- Share the location of all archive log destinations with all nodes
- Have each node back up its own archive log
- Move the archive logs to one node and back them up

You can use RMAN to implement the first and second solutions and operating system utilities to implement the third.

Backing Up Archive Logs with RMAN

If you share all archive logs with all nodes of a cluster, backup is very easy and can be executed from any node because every node can read all the logs. In the example below, node 1 backs up all redo logs of all nodes. Make sure that the directories are configured for sharing as described in the *Oracle8i Parallel Server Setup and Configuration Guide.*

```
rman TARGET INTERNAL/sys@node1 catalog rman/rman@rman

  RUN {
         ALLOCATE CHANNEL t1 type ’sbt_tape’ FORMAT ’al_t%t_s%s_p%p’;
```

```
        SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
        BACKUP ARCHIVELOG ALL DELETE INPUT;
        RELEASE CHANNEL t1;
        }
```

With the ALTER SYSTEM ARCHIVE LOG CURRENT statement, you force all nodes to back up their current log files.

If you do not share all archive logs, you can back up the logs locally on every node. In case of recovery, however, you need to have access from the node on which you begin recovery to all the archive logs on all nodes. For this reason Oracle recommends using a media management system that supports archiving over the network or shared directory services to simplify restoring log files. The following RMAN script starts the local backup of all nodes using the CONNECT and LIKE clauses.

Note: Oracle recommends using the LIKE clause instead of THREAD so that one instance can archive logs on behalf on another thread when the other thread is down. As well, the archive log destinations of the various instances must be different so that LIKE can distinguish them from one another.

```
rman TARGET internal/sys@node1 catalog rman/rman@rman

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_n1_t%t_s%s_p%p'
        CONNECT internal/sys@node1;
        SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
        BACKUP ARCHIVELOG LIKE '%/arch1/%' delete input;
        RELEASE CHANNEL t1;
        }

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_n2_t%t_s%s_p%p'
        CONNECT internal/sys@node2;
        BACKUP ARCHIVELOG LIKE '%/arch2/%' DELETE INPUT;
        RELEASE CHANNEL t1;
        }

  RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_n3_t%t_s%s_p%p'
        CONNECT internal/sys@node3;
```

```
         BACKUP ARCHIVELOG LIKE '%/arch3/%' DELETE INPUT;
         RELEASE CHANNEL t1;
         }
```

Back up all the archive logs from one node into one backup archive instead of archiving them from each node separately. This makes it easier to find all backups during recovery. If you do not use shared directories to back up and restore archive logs, copy or move them using operating system tools. You can easily create scripts to do this job before backing up or restoring the logs.

To copy all archive logs to the local directories on node 1 use a script similar to the following:

```
#!/bin/sh
sqlplus system/manager@node1 @switchlog.sql
rcp node2:/u01/app/oracle/product/815/admin/ops/arch2/*
/u01/app/oracle/product/815/admin/ops/arch2
rcp node3:/u01/app/oracle/product/815/admin/ops/arch3/*
/u01/app/oracle/product/815/admin/ops/arch3
```

The `switchlog.sql` script is used to make sure to get all necessary log files for recovery. It looks like this:

```
#!/bin/sh
ALTER SYSTEM ARCHIVE LOG CURRENT;
EXIT
```

To back up the archived logs from node 1 using RMAN, the command is similar to the example on page 13-8 except that the ALTER SYSTEM ARCHIVE LOG CURRENT statement is executed from the shell script:

```
rman TARGET internal/sys@node1 catalog rman/rman@rman

  RUN {
         ALLOCATE CHANNEL t1 TYPE 'sbt_tape' FORMAT 'al_t%t_s%s_p%p';
         BACKUP ARCHIVELOG ALL DELETE INPUT;
         RELEASE CHANNEL t1;
         }
```

Restoring Archive Logs with RMAN

If RMAN has concurrent access to all backups, it automatically restores all necessary archive logs from previous backups for recovery. In Oracle Parallel Server environments, the restore procedure varies depending on the option you used to back up the archive logs.

If you share archive log directories, you can change the destination of the automatic restoration of archive logs with the SET clause to restore the files to a local directory of the node from where you begin recovery.

To restore the USERS tablespace from node 1, use an RMAN command syntax similar to the following:

```
rman TARGET internal/sys@node1 catalog rman/rman@rman

  run {
         allocate channel t1 type ’sbt_tape’;
         set archivelog destination to
’/u01/app/oracle/product/815/admin/ops/arch1’;
         recover tablespace users;
         sql ’alter tablespace users online’;
         release channel t1;
         }
```

If you backed up each node’s log files using a central media management system, you can use the RMAN AUTOLOCATE option of the SET command. If you use several channels for recovery, RMAN asks every channel for the required file if it does not find it in the first one. This feature allows you to recover a database using the local tape drive on the remote node:

```
rman TARGET internal/sys catalog rman/rman@rman
  RUN {
         ALLOCATE CHANNEL t1 type ’sbt_tape’ parms ’ENV=(NSR_CLIENT=node1)’;
         ALLOCATE CHANNEL t2 type ’sbt_tape’ parms ’ENV=(NSR_CLIENT=node2)’;
         ALLOCATE CHANNEL t3 type ’sbt_tape’ parms ’ENV=(NSR_CLIENT=node3)’;
         SET AUTOLOCATE ON;
         RECOVER TABLESPACE users;
         SQL ’ALTER TABLESPACE users ONLINE’;
         RELEASE CHANNEL t1;}
```

If you backed up the logs from each node without using a central media management system, you must first restore all the log files from the remote nodes and move them to the host from which you will start recovery. This means you must perform recovery in three steps:

1. Restore the datafiles.
2. Restore the archive logs.
3. Begin recovery.

```
rman target internal/sys catalog rman/rman@rman
RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' connect internal/sys@node1;
        RESTORE TABLESPACE users;
        RELEASE CHANNEL t1;
        }

        RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' connect internal/sys@node2;
        RESTORE ARCHIVELOG
           # this line is optional if you don't want to restore ALL archive
logs:
           FROM TIME "to_date('05.09.1999 00:00:00','DD.MM.YYYY HH24:Mi:SS')"
           LIKE '%/2_%';
        RELEASE CHANNEL t1;
        }
RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape' connect internal/sys@node3;
        RESTORE ARCHIVELOG
           # this line is optional if you don't want to restore ALL archive
logs:
           FROM TIME "to_date('05.09.1999 00:00:00','DD.MM.YYYY HH24:Mi:SS')"
           like '%/3_%';
        RELEASE CHANNEL t1;
        }

        EXIT

rcp node2:/u01/app/oracle/product/815/admin/ops/arch2
/u01/app/oracle/product/815/admin/ops/arch2
rcp node3:/u01/app/oracle/product/815/admin/ops/arch2
/u01/app/oracle/product/815/admin/ops/arch2

rman TARGET internal/sys catalog rman/rman@rman

        RUN {
        ALLOCATE CHANNEL t1 TYPE 'sbt_tape';
ALLOCATE CHANNEL d1 type disk;
RECOVER TABLESPACE users;
SQL 'ALTER TABLESPACE USERS ONLINE';
}
```

Note: In the recover step, there is an 'sbt_tape' channel allocated so that the archivelogs generated on node 1 will be automatically restored.

If you moved all archive logs to one node to back them up, recovery is as easy as recovery using shared directories. To make sure you have all the log files, copy all remote log files with your shell script as in this example:

```
/rcp_all_logs.sh
rman TARGET internal/sys@node1 catalog rman/rman@rman
RUN {
        ALLOCATE CHANNEL t1 type 'sbt_tape' format 'al_t%t_s%s_p%p';
        BACKUP ARCHIVELOG ALL DELETE INPUT;
        RELEASE CHANNEL t1;
        }
```

Checkpoints and Log Switches

This section discusses:

- Checkpoints
- Forcing a Checkpoint
- Forcing a Log Switch
- Forcing a Log Switch on a Closed Thread

Checkpoints

Oracle performs checkpointing automatically on a consistent basis. Checkpointing requires that Oracle write all dirty buffers to disk and advance the checkpoint.

See Also: *Oracle8i Designing and Tuning for Performance* for more information about checkpoints.

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 data files on disk.

The GLOBAL clause 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 requiring 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 CONNECT statement.

Note: You need the ALTER SYSTEM privilege to force a checkpoint.

See Also: "Setting and Connecting to Instances" on page 4-3 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. This can be done 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 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.

When all instances to perform forced log switches, it is known as a "global log switch." To do this, use the SQL statement ALTER SYSTEM ARCHIVE LOG CURRENT omitting 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.

Use the INSTANCE FORCE LOG SWITCH clause for each instance; there is no global option for forcing a log switch. 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.

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 SQL statement ALTER SYSTEM with the ARCHIVE LOG clause. 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 describes backup operation issues in Oracle Parallel Server. It covers the following topics:

- Open and Closed Database Backups
- RMAN Backup Issues
- Operating System Backup Issues

Open and Closed Database Backups

You can perform all backup operations from any node of an Oracle Parallel Server. Open backups allow you to back up all or part of the database while it is running. Users can update data in any part of the database during an open backup. With Oracle Parallel Server you can make open backups of multiple tablespaces simultaneously from different nodes. An open backup includes copies of one or more data files 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 you use the operating system, closed backups are done while the database is closed. When you use RMAN, an instance must be started and mounted, but not open, to perform closed backups. Before making a closed backup, shut down *all* instances of your Oracle 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 data files 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 closed, whole backup.**

Never erase, reuse, or destroy archived redo log files until completing another whole backup, or preferably two whole backups, in either open or closed mode.

See Also:

- *Oracle8i Backup and Recovery Guide*.
- *Oracle8i Concepts*.

Online Backups and Oracle Parallel Server

Online backups in Oracle Parallel Server are efficient because they do not use the cache. This means you can run online backups from a single instance in the cluster and not experience pinging.

Because backups use primarily CPU resources, so you can make use of the less busy instances. However, you should monitor disk usage to ensure that the I/O is not being saturated by the backup. If the I/O is saturated by the backup, it may adversely affect the online users.

Note: Using the ALTER TABLESPACE ... BEGIN BACKUP statement generates extra redo logs.

RMAN Backup Issues

This section describes the following RMAN backup issues:

- Preparing for Snapshot Control Files in RMAN
- Performing an Open Backup Using RMAN
- Node Affinity Awareness

Preparing for Snapshot Control Files in RMAN

In Oracle Parallel Server, you must prepare for snapshot control files before performing backups using RMAN.

Any node making a backup may need to create a snapshot control file. Therefore, on all nodes used for backup, ensure the existence of the destination directory for such a snapshot control file.

For example, to specify that the snapshot control file should be written to the file `/oracle/db_files/snapshot/snap_prod.cf`, 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 data files in Oracle Parallel Server will be shared across all nodes in the cluster.

Performing an Open Backup Using RMAN

If you are also backing up archive logs, then issue an ALTER SYSTEM ARCHIVE LOG CURRENT statement after the backup is complete. This ensures that you have all redo data to make the files in your backup consistent.

The following sample script distributes data file 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);
```

See Also : *Oracle8i Recovery Manager User's Guide and Reference* for complete information on open backups using RMAN.

Node Affinity Awareness

On some cluster platforms, certain nodes of the cluster have faster access to some data files than to other data files. RMAN automatically detects this type of affinity. When deciding which channel will back up a particular data file, RMAN gives preference to channels allocated at nodes with affinity to that data file. To use this feature, allocate RMAN channels at the various nodes of the cluster that have affinity to the data files being backed up.

For example:

```
RUN
{
  ALLOCATE CHANNEL CH1 TYPE 'SBT_TAPE' CONNECT '@INST1';
  ALLOCATE CHANNEL CH2 TYPE 'SBT_TAPE' CONNECT '@INST2';
  ...
}
```

See Also: *Oracle8i Backup and Recovery Guide* for more information about the CONNECT clause of the ALLOCATE statement.

Operating System Backup Issues

This section discusses the following operating system 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 data files...

....COPY COMPLETED...

ALTER TABLESPACE TABLESPACE END BACKUP;/* INSTANCE y */
Statement processed.
```

Note: If you do not issue the ALTER TABLESPACE ... BEGIN BACKUP statement, or if processing does not complete before an operating system backup of the tablespace begins, then the backed up data files 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 clause has no effect on user access to tablespaces.

For an open backup to be usable for complete or incomplete media recovery, retain all archived redo logs spanning the period of time between the execution of the BEGIN BACKUP statement 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:** *Oracle8i SQL Reference* for a description of the BEGIN BACKUP and END BACKUP clauses of the ALTER TABLESPACE statement.

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 Oracle Parallel Server.

1. Before starting the open backup, issue the ALTER SYSTEM ARCHIVE LOG CURRENT statement.

   This switches and archives the current redo log file for *all* threads in your Oracle Parallel Server environment, including threads that are not currently up.
2. Issue the ALTER TABLESPACE *tablespace* BEGIN BACKUP statement.
3. Wait for the ALTER TABLESPACE statement to successfully complete.
4. In the operating-system environment, issue the appropriate statements to back up the data files for the tablespace.
5. Wait for the operating-system backup to successfully complete.
6. Issue the ALTER TABLESPACE *tablespace* END BACKUP statement.
7. Back up the control files with ALTER DATABASE BACKUP CONTROLFILE TO *filename*.

For added safety, back up the control file to a trace file with the ALTER DATABASE BACKUP CONTROLFILE TO TRACE NORESETLOGS statement, 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 back to the "end backup" marker.

14

Recovering the Database

This chapter describes Oracle recovery features on Oracle Parallel Server. It covers the following topics:

- Recovery from Instance Failure
- Recovery from Media Failure
- Parallel Recovery

> **See Also:** *Oracle8i Backup and Recovery Guide* for general information about Oracle recovery.

Three Types of Recovery

This chapter discusses three types of recovery:

- Recovery from Instance Failure
- Recovery from Media Failure
- Parallel Recovery

Recovery from Instance Failure

Instance failure occurs when a software or hardware problem prevents an instance from continuing work. 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 Rollback
- Access to Data Files for Instance Recovery
- Steps 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 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: *Oracle8i Backup and Recovery Guide.*

Single-Node Failure

Oracle Parallel Server 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 14-2.

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.

> **See Also:** *Oracle8i Parallel Server Concepts* for more information about application failover.

Multiple-Node Failure

As long as one instance continues running, its SMON process performs instance recovery for any other instances that fail.

If all instances of an Oracle 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 Oracle 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 Oracle. Fast-start checkpointing occurs continuously, advancing the checkpoint as Oracle writes 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 the duration of the roll forward phase of Fast-start checkpointing. Oracle automatically adjusts the checkpoint write rate to meet the specified roll-forward limit while issuing the minimum number of writes.

> **See Also:** *Oracle8i Designing and Tuning for Performance* for details on how to do this.

Fast-Start Rollback

The rollback phase of system fault recovery in Oracle 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 Parallel Server Concepts* for more information on Fast-start rollback.

Access to Data Files for Instance Recovery

An instance performing recovery for another instance must have access to all online data files that the failed instance was accessing. When instance recovery fails because a data file fails verification, the instance that attempted to perform recovery does not fail but a message is written to the alert log file.

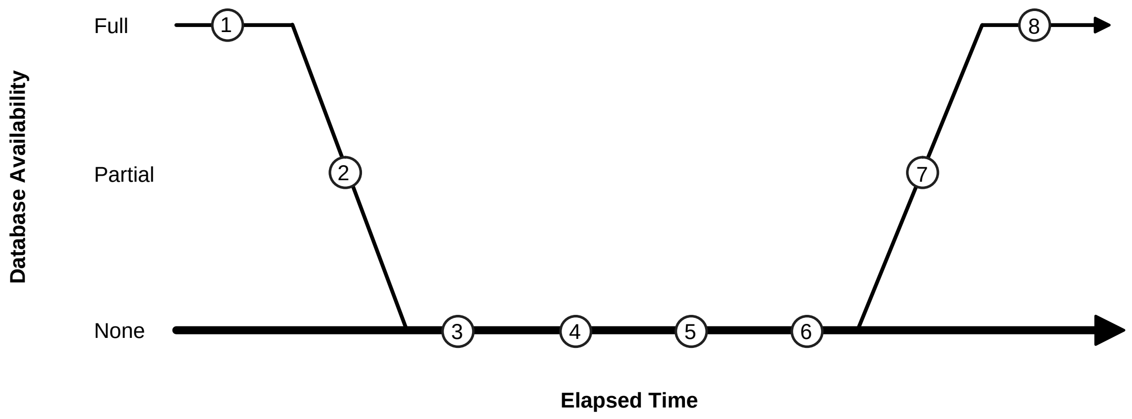
After you correct the problem that prevented access to the database files, use the SQL statement ALTER SYSTEM CHECK DATAFILES to make the files available to the instance.

See Also: "Datafiles" on page 6-2.

Steps of Oracle Instance Recovery

Figure 14–1 illustrates the degree of database availability during each step of Oracle instance recovery.

Figure 14–1 Steps of Oracle Instance Recovery



The steps involved in recovery are:

1. Oracle Parallel Server 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 threads 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 step 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 data files, regardless of whether you are using high or low granularity 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 DLM for user data

Oracle can read buffers already in the cache with the correct global lock because this does not involve any I/O or lock operations.

The transitional state of the buffer cache begins at the conclusion of the initial lock scan step 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

Media failure occurs when the storage medium for Oracle files is damaged. This usually prevents Oracle from reading or writing data after a media failure resulting in the loss of one or more database files, use backups of the data files to recover the database. If you are using Recovery Manager (RMAN), you may 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: *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 14–1 shows the status of the database that is required to recover particular database objects.

Table 14–1 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 data file.	The database can be open with the data file offline, or the database can be mounted but not opened by any instance. (For a data file in the SYSTEM tablespace, the database must be mounted but not open.)

You can recover multiple data files 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 data files in shared mode. Do this using the RECOVER TABLESPACE or RECOVER DATAFILE statements.

You can use the RECOVER DATABASE statement to recover a database that is mounted in shared mode, but not open. Only one instance can issue this statement in Oracle Parallel Server.

Note: The recommended method of recovering a database is to use RMAN. Oracle does not recommend use of the SQL statement ALTER DATABASE RECOVER.

Incomplete Media Recovery

You can perform incomplete media recovery while the database is mounted in shared or exclusive mode providing it is not opened by an instance. Do this using the following database recovery options:

With RMAN use one of the following clauses with the SET statement before restoring and recovering:

- UNTIL CHANGE *integer*
- UNTIL TIME *date*
- UNTIL LOGSEQ *integer* THREAD *integer*

With operating system utilities, restore your backups and then use one of the following clauses with the RECOVER DATABASE statement:

- UNTIL CANCEL
- UNTIL CHANGE *integer*
- UNTIL TIME *date*

See Also: *Oracle8i Backup and Recovery Guide* for more information on the use of the SET and RECOVER DATABASE statements.

Restoring and Recovering Redo Log Files

Media recovery of a database accessed by Oracle Parallel Server may require the simultaneous opening of multiple archived log files. 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 RMAN

RMAN automatically restores and applies the archive logs required. By default, RMAN 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 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 using the following platform-specific methods.

Making Archive Logs Readable by All Nodes For detailed procedures on how to configure this, refer to the *Oracle8i Parallel Server Setup and Configuration Guide.*

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, you receive these messages but not the prompt. Redo log files may be required for each enabled thread in Oracle Parallel Server. 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 Oracle Parallel Server, 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 RMAN 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 RMAN

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, because all previous backups are invalidated. This step is not shown in this example.

The SET UNTIL statement 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, RMAN 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, data file 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 RMAN 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 data files' new locations.
- Recovers the data files by either using a combination of incremental backups and redo, or just redo

  RMAN 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:

Start SQL*Plus as follows:

```
CONNECT scott/tiger AS SYSDBA
```

Oracle responds with:

```
Connected.
```

Enter the following STARTUP syntax:

```
STARTUP NOMOUNT RESTRICT
```

Start RMAN and run 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 SQL*PLus.
3. Connect as SYSDBA.
4. Start and mount the database with the STARTUP MOUNT statement.
5. Initiate an incomplete recovery using the RECOVER statement with the appropriate UNTIL clause.

   The following statement 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 statement.
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 computed 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 data files on several disks are being recovered concurrently.

For RMAN, the restore and application of incremental backups are parallelized using channel allocation. The RECOVERY_PARALLELISM parameter determines the number of concurrent processes that participate in recovery. Setting RECOVERY_PARALLELISM to 0 or 1 invokes serial recovery.

Parallel Recovery Using RMAN

With RMAN's RESTORE and RECOVER statements, Oracle can automatically parallelize all three stages of recovery.

Restoring Data Files: When restoring data files, the number of channels you allocate in the RMAN recover script effectively sets the parallelism RMAN 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 redo logs using a specific number of parallel processes as determined by your setting for the RECOVERY_PARALLELISM parameter.

The RECOVERY_PARALLELISM initialization parameter specifies the number of redo application server processes participating in instance or media recovery. During parallel 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 data files. A value of 0 or 1 indicates that Oracle performs recovery serially. The value of this parameter cannot exceed the value of the PARALLEL_MAX_SERVERS parameter.

Parallel Instance Recovery

Parallel execution can also improve recovery processing. To use parallel execution for recovery, the parallel execution processes must be running when the instance starts up.

Set the PARALLEL_MIN_SERVERS parameter to establish the number of parallel execution servers available for parallel recovery. You can do this even if you do not intend to use parallel execution for the rest of your Oracle processing. Use the PARALLEL_MAX_SERVERS parameter to set a limit on the number of parallel execution processes available for recovery.

Media Recovery

The PARALLEL clause of the RECOVER DATABASE statement determines the degree of parallelism in media recovery. Media recovery uses the value for RECOVERY_PARALLELISM as a default degree of parallelism if the value for this parameter is non-zero and if you do not supply a value for RECOVERY_PARALLELISM in the RECOVER DATABASE statement. You can override this degree of parallelism with the PARALLEL clause of the RECOVER DATABASE statement.

For instance recovery, set the RECOVERY_PARALLELISM parameter to equal the number of parallel execution servers that you want to be available to assist SMON during recovery.

Parallel Recovery Using Operating System Utilities

You can parallelize instance and media recovery two ways by:

- Setting the RECOVERY_ PARALLELISM Parameter
- Specifying RECOVER Statement Options

The Oracle Parallel 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 data files. 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 data files. 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 Statement Options

When you use the RECOVER statement 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
- Oracle system-specific documentation for more information on the allocation of recovery processes to instances

Fast-Start Parallel Rollback in Oracle Parallel Server

Setting the initialization file 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 set to FALSE, Fast-start parallel rollback is disabled.

If the value for FAST_START_PARALLEL_ROLLBACK is set to LOW, the number of processes used for Fast-start rollback is twice the value of 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 Oracle Parallel Server, 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.

Disaster Protection Strategies

You can protect Oracle Parallel Server systems against disasters by using standby databases, the Primary / Secondary Instance feature, and by following high availability practices for Oracle Parallel Server.

To simplify the administration of standby databases, consider using a managed standby database.

See Also:

- *Oracle8i Standby Database Concepts and Administration* for details about the managed standby database feature.
- *Oracle8i Parallel Server Concepts* for more information on high availability and the Primary / Secondary Instance feature.

Part VI

Oracle Parallel Server Reference

Part Six contains reference material about Oracle Parallel Server. It contains the following appendix:

- Appendix A, "A Case Study in Parallel Server Database Design"

A

A Case Study in Parallel Server Database Design

This appendix describes a case study that presents a methodology for designing systems optimized for Oracle Parallel Server.

Case Study Overview

The case study presented in this appendix provides techniques for designing new applications for use with Oracle Parallel Server. You can also use these techniques to evaluate existing applications and determine how well suited they are for migration to Oracle Parallel Server.

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

This case study assumes you have made an initial database design. To optimize your design for Parallel Server, 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 high or low granularity locking.
7. Implement and tune your design.

See Also: Part III, "Oracle Parallel Server Design and Deployment", for detailed information on this methodology.

Case Study: From Initial Database Design to Oracle Parallel Server

This case study demonstrates analytical techniques in practice. Although your applications will differ, this example helps you to understand the process. The topics in this section are:

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

"Eddie Bean" Catalog Sales

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

- Order entry
- Accounts payable
- Accounts receivable

Tables

Tables from the Eddie Bean database include:

Table A–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 occur 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 budgets, and are deleted once they go into actuals.

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

Analyzing Access to Tables

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

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

Table Access Analysis Worksheet

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

Table A–2 Table Access Analysis Worksheet

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

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

Estimating Volume of Operations

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

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

Calculating I/Os per Operation

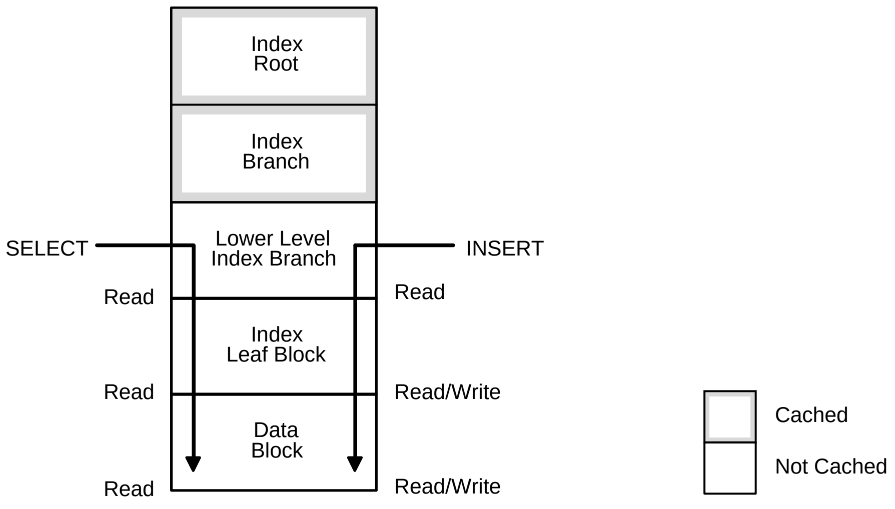
For each value in the Operations column, calculate the number of I/Os that will be generated using a worst-case scenario.

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 A–1 illustrates read and write access to data in a large table in which two levels of the index are not in the buffer cache and only a high level index is cached in the System Global Area.

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



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

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

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

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

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

One UPDATE in this example entails seven I/Os:

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

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

I/Os 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 A–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 A–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 A–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 A–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 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 Oracle Parallel Server application.

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

Case Study: Table Access Analysis

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

Table A–5 Case Study: Table Access Analysis Worksheet

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

You can make the following conclusions from the data in 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.

Analyzing Transaction Volume by Users

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

- Transaction Volume Analysis Worksheet
- Case Study: Transaction Volume Analysis

Transaction Volume Analysis Worksheet

For each table with a high volume of write access, analyze the transaction volume per day for each type of user.

Note: For read-only tables, you do *not* need to analyze transaction volume by user type.

Use worksheets like the one in Table A–6:

Table A–6 Transaction Volume Analysis Worksheet

Table Name:									
		Daily Transaction Volume							
		Read Access		Write Access					
		Select		Insert		Update		Delete	
Type of User	No.Users	Operations	I/Os	Operations	I/Os	Operations	I/Os	Operations	I/Os

Begin by estimating the volume of transactions by each type of user and then calculate the number of I/Os required.

Case Study: Transaction Volume Analysis

The following tables show transaction volume analysis of the three tables in the case study that have high write access levels: ORDER_HEADER, ORDER_ITEMS, and ACCOUNTS_PAYABLE.

ORDER_HEADER Table

Table A–7 shows rough estimates for values in the ORDER_HEADER table in the case study.

Table A–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

You can make the following conclusions from the data in 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 A–8 shows rough estimates for values in the ORDER_ITEMS table in the case study.

Table A–8 Case Study: Transaction Volume Analysis: ORDER_ITEMS Table

Table Name: ORDER_ITEMS									
		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**
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 A–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 A–9 Case Study: Transaction Volume Analysis: ACCOUNTS_PAYABLE Table

Table Name: ACCOUNTS_PAYABLE									
		Daily Transaction Volume							
		Read Access		**Write Access**					
Type of	**No.**	**Select**		**Insert**		**Update**		**Delete**	
User	**Users**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**	**Operations**	**I/Os**
Order entry clerk	25	0	0	0	0	0	0	0	0
Accounts payable clerk	5	200	600	50	250	20	140	0	0
Accounts receivable clerk	5	0	0	0	0	0	0	0	0
Shipping clerk	4	0	0	0	0	0	0	0	0
Sales manager	2	0	0	0	0	0	0	0	0
Financial analyst	2	30	90	0	0	0	0	0	0

You can make the following conclusions from the data in this table:

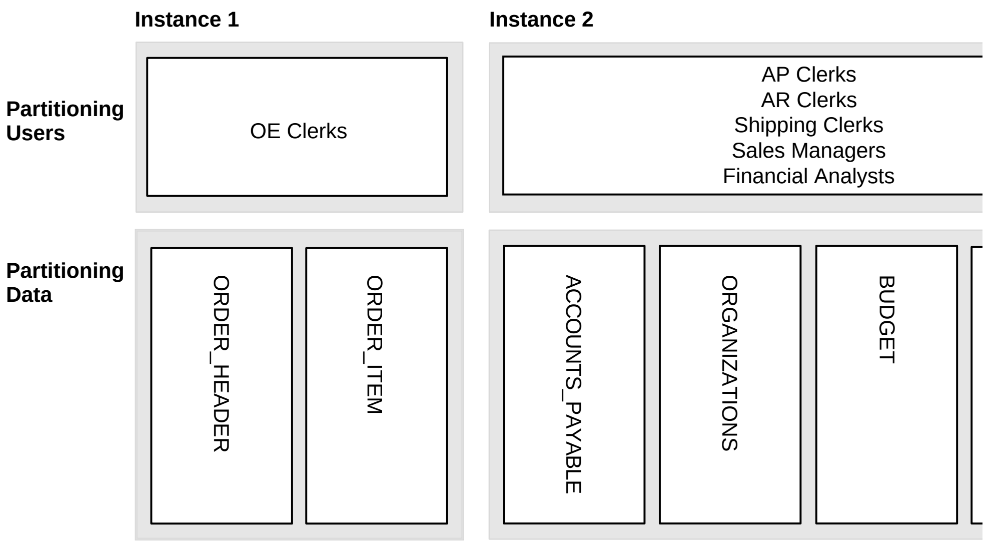
- Accounts payable clerks send about 50 purchase orders per day to suppliers. These clerks are the only users who change the data in this table.
- Financial analysts occasionally study the information.

Deletes are performed as a maintenance operation, so you do not need to consider them in this analysis.

Case Study: Initial Partitioning Plan

In the case study, the large number of order entry clerks doing heavy insert activity on the ORDER_HEADER and ORDER_ITEM tables should not be separated across machines. You should concentrate these users on one node along with the two tables they use most. A good starting point is to set aside one node for the OE clerks, and one node for all other users as illustrated in Figure A–2.

Figure A–2 Case Study: Partitioning Users and Data



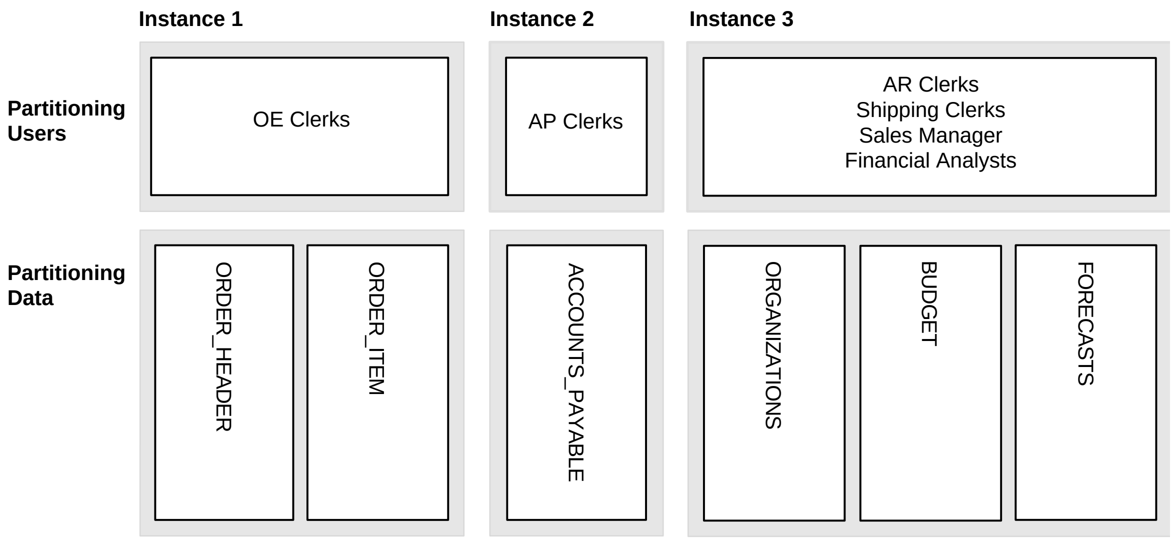
This system is probably well balanced across nodes. The database intensive reporting done by financial analysts takes a good deal of system resources, whereas the transactions run by the order entry clerks are relatively simple.

Attempting to use 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 onto a separate instance as shown in Figure A–3.

Figure A–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 moving between instances.

Based on this analysis, you have two design options as described under the following headings.

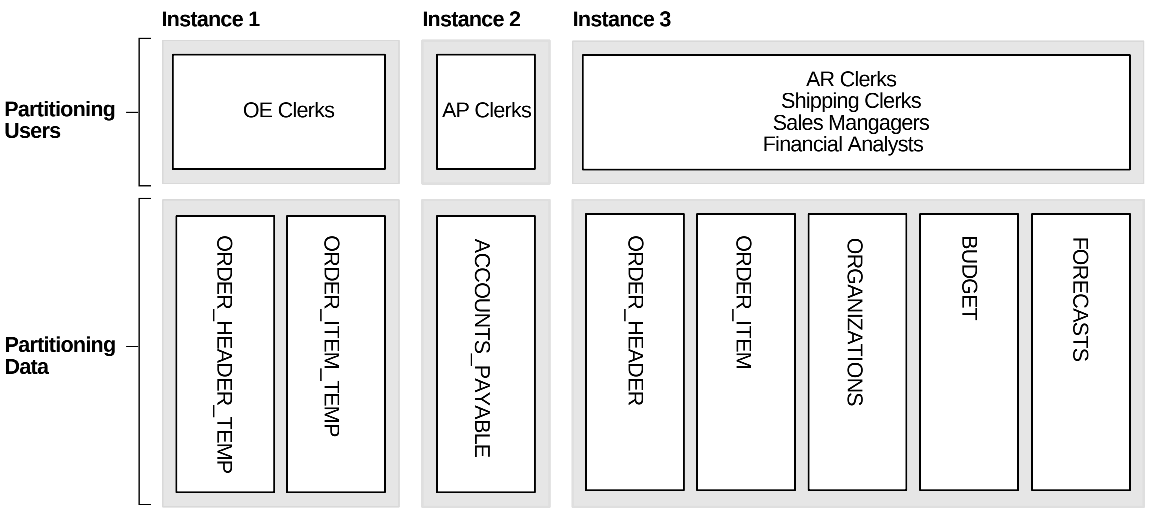
Design Option 1

You can set up your system as shown in Figure A–3 with all order entry clerks on one instance to minimize contention for exclusive PCM locks on the tables. This allows sales managers and financial analysts to get up-to-the-minute information. Since they do want data that is predominantly historical, there should not be too much contention for current records.

Design Option 2

Alternatively, you could implement a separate temporary table for ORDER_ITEM/ ORDER_HEADER. This table is only for recording new order information. Overnight, you could incorporate changes into the main table against which all queries are performed. This solution would work well if it is not required that financial analysis have current data. This is probably an acceptable solution only if they are primarily interested in looking at historical data. This would not be appropriate if the financial analysts needed up-to-the-minute data.

Figure A–4 Case Study: Partitioning Users and Data: Design Option 2



Partitioning Indexes

You need to consider index partitioning if multiple nodes in your system are inserting into the same index. In this situation, you must ensure that different instances insert into different points within the index.

Note: This problem is avoided in the Eddie Bean case study because application and data usage are partitioned.

See Also:

- "Creating Free Lists for Indexes" on page 8-6 for tips on using free lists, free list groups, and sequence numbers to avoid contention on indexes.
- *Oracle8i Concepts* for recommendations on how to physically partition a table and an instance to avoid the use of free list groups.

Implementing High or Low Granularity Locking

For many applications, the DBA needs to decide whether to use high or low granularity locking for particular database files.

You should design for the worst case scenario that would use high granularity 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 low granularity locking.

Begin with an analysis at the database level. You can use a worksheet like the one shown in Table A–10:

Table A–10 Worksheet: Database Analysis for High or Low Granularity Locking

Block Class	Relevant Parameter(s)	Use High or Low Granularity Locking?

Next, list the files and database objects in a worksheet like the one shown in Table A–11. Decide which locking mode to use for each file.

Table A–11 Worksheet: When to Use High or Low Granularity Locking

Filename	Objects Contained	Use High or Low Granularity Locking?

See Also: Chapter 9, "Setting Instance Locks" for more information about applying locks to data files.

Implementing and Tune Your Design

Up to this point, you conducted an analysis using estimated figures. To finalize your design you must now either prototype the application or actually implement it. By observing the actual system, you can tune it further.

To do this, try the following techniques:

- Identify blocks that are being pinged and determine where contention exists.
- Consider moving users from one instance to another to reduce 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: *Oracle8i Designing and Tuning for Performance*.

Index